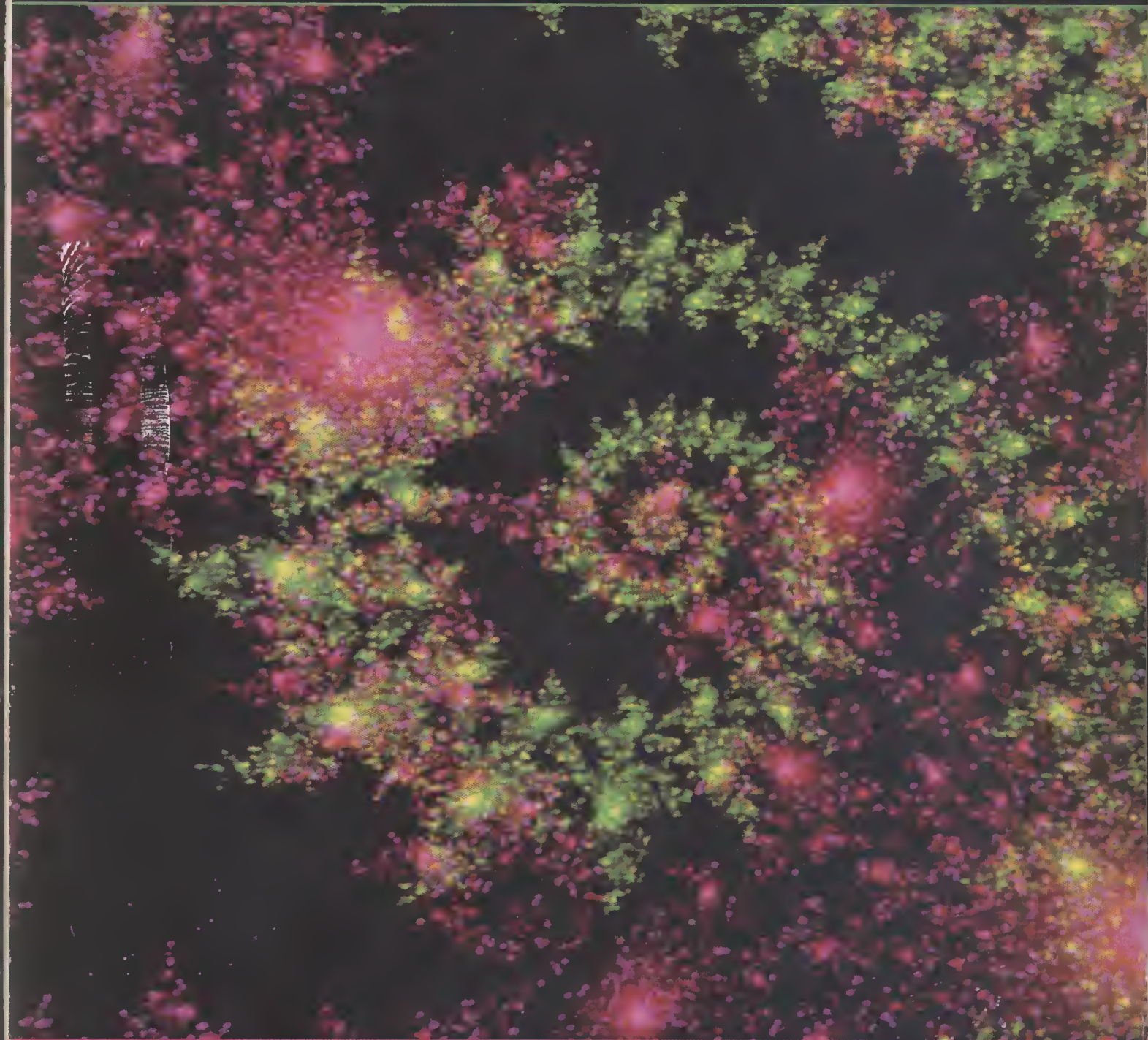


JANUARY 1987

COMPUTER



New AI Systems:
HARNESSING THE COMBINATORIAL EXPLOSION

 COMPUTER SOCIETY

 THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.

THEY'RE HERE! FREE! 1987 Handbooks & Encyclopedias



Get Yours Now!

OMEGA
ENGINEERING, INC.
(203) 358-1660

One Omega Drive, Box 4047
Stamford, CT 06907

Reader Service Number 1

COMPUTER

January 1987

Volume 20, Number 1

ARTICLES

10 Guest Editor's Introduction: New Computers for Artificial Intelligence Processing

Benjamin W. Wah

New computer architectures can be used to improve efficiency in the processing of some time-consuming AI tasks, but cannot overcome the combinatorial complexity of AI processing.

19 Computer Architectures for Artificial Intelligence Processing

Kai Hwang, Joydeep Ghosh, and Raymond Chowkwanyun

This article assesses a new generation of intelligent multiprocessors and multicomputers with respect to current AI technologies, advanced computer architectures, and application demands.

30 Software Development Support for AI Programs

C. V. Ramamoorthy, Shashi Shekhar, and Vijay Garg

In spite of the widespread commercial use of expert systems, vendors cannot guarantee the quality of the programs.

43 Symbolics Architecture

David A. Moon

This architecture enables rapid development and efficient execution of large, ambitious applications. An unconventional design avoids trading off safety for speed.

55 The Architecture of FAIM-1

Judy M. Anderson, William S. Coates, Alan L. Davis, Robert W. Hon, Ian N. Robinson, Shane V. Robison, and Kenneth S. Stevens

FAIM-1 can be scaled to provide two to three orders of magnitude improvement over conventional AI machines with minimal inconvenience to programmers.

67 What Price Smalltalk?

David Ungar and David Patterson

Are reduced instruction set computers good for languages like Lisp and Smalltalk? With the addition of a few simple hardware changes and software techniques—yes.

75 Initial Performance of the DADO2 Prototype

Salvatore J. Stolfo

A 1023-processor prototype named DADO2 has successfully demonstrated performance improvements for several important problems and has led to a number of avenues of future research.

85 Applications of the Connection Machine

David L. Waltz

This massively parallel system is easily programmable, applicable to a variety of applications, and can be scaled to large problems by the addition of more processors.

100 Connectionist Architectures for Artificial Intelligence

Scott E. Fahlman and Geoffrey E. Hinton

Massively parallel networks of simple neuron-like processing elements may hold the key to some important aspects of intelligence not captured by existing AI technology on serial machines.

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the IEEE Computer Society. IEEE Headquarters: 345 East 47th St., New York, NY 10017; IEEE Computer Society West Coast Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-8380. Annual subscription included in society member dues. Nonmember prices: available upon request. Single copy prices: members \$7.50; nonmembers \$15.00. This magazine is also available in microfiche form.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Second class postage is paid at New York, New York, and at additional mailing offices.

Copyright and reprint permissions: Copyright © 1987 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons: (1) those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress Street, Salem, MA 01970. (2) pre-1978 articles without fee. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to True Seaborn, Editor and Publisher, *Computer*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720. All rights reserved. Copyright © 1987 by The Institute of Electrical and Electronics Engineers, Inc.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space considerations. *Computer* subscribes to the Computer Press Association's code of professional ethics.



Computers for AI Applications

DEPARTMENTS

9 Letters to the Editor:

SDI should not be designed; Response to
"LisaLearning" article

114 Book Reviews:

Expert Systems and Fuzzy Systems;
Fundamentals of Digital Electronics

115 Open Channel:

Does more performance mean less utility?

116 Update:

COPP committee report; glossary published;
FJCC; AFIPS seeks nominees; Eckert-
Mauchly nominations

121 New Products

124 New Literature

126 IC Announcements

127 Microsystem Announcements

128 Call for Papers

129 Calendar

130 Career Opportunities

144 Advertiser/Product Index

144A Reader Service Card



On the cover

The fractal on our cover symbolizes the combinatorial explosion characteristic of AI processing. For details and other examples of fractals, see "About the Cover," p. 13. The fractals were all supplied by Art Matrix of Ithaca, New York. The cover was designed by Jay Simpson.

In the next issue

Future FAA systems

COMPUTER

10662 Los Vaqueros Circle
Los Alamitos, CA 90720

Editor-in-Chief: Michael C. Mulder, University of Portland

Editorial Board

Dharma P. Agrawal, North Carolina State University
Jim Haynes, University of California, Santa Cruz
Ronald G. Hoelzeman, University of Pittsburgh
Franklin F. Kuo, SRI International

Francis P. Mathur,

California State Polytechnic University, Pomona

K. M. Mohiuddin, IBM Corporation

Arthur V. Pohm, Iowa State University

Ralph Preiss, IBM Corporation

C. V. Ramamoorthy, University of California, Berkeley

Sallie Sheppard, Texas A&M University

Harry Strong, Mitre Corporation

Wing N. Toy, AT&T Bell Laboratories

Herbert Weber, University of Dortmund

Helen M. Wood, National Bureau of Standards

Magazine Advisory Committee

Dennis R. Allison (chair), Dharma P. Agrawal, Vishwani D. Agrawal,
James J. Farrell III, Lansing Hatfield, Michael C. Mulder, David Pessel,
True Seaborn, Bruce D. Shriver

Administration

Editor and Publisher: True Seaborn

Assistant Publisher: Gary L. Carter

Assistant to the Publisher: Pat Paulsen

Editorial Staff

Managing Editor: Marilyn Potes

Issue Editor: Louise Anderson

Assistant Editors: Nancy Hays, Thomas Szalkiewicz

Contributing Editor: Ware Myers

Production

Design/Production: Miriam Wiegel

Art Director: Jay Simpson

Production Supervisor: David Gaines

Circulation/Advertising

Circulation Manager: Christina Champion

Advertising Coordinator: Carole Porter

Submissions: Send six double-spaced copies of articles and special-issue proposals to Bruce D. Shriver, IBM T.J. Watson Research Center, PO Box 704, H0-B04A, Yorktown Heights, NY 10598; (914) 789-7626. For electronic submission, Shriver's addresses are Compmail + , b.shriver; CSnet, Shriver@ibm.com; Vnet, shriver at yktvmh.



YOU can't afford to miss COMPCON Spring '87!
The year's only broad-based computing conference
by the world's largest computer society.

READING conference PROCEEDINGS IS NO SUBSTITUTE for hearing and interacting with speakers, audiences and peers!

KEY HOT TOPICS:

- EMERGING NEW MINISUPERS • EMERGING NEW SOFTWARE TRENDS
- COMPUTER SECURITY • NETWORKING & ASSOCIATED FILE STRUCTURES

PROGRAM HIGHLIGHTS: 41 SESSIONS, 120 SPEAKERS

(Subject to adjustment)

PLENARY TALKS:

Tuesday Morning, February 24, 1987

"Future Directions for High-end Work Stations"
Dr. Andries Van Dam, Brown University.

Wednesday Afternoon, February 25, 1987

"Trends in Knowledge Processing: From Expert
Systems to Intelligent Systems Engineering"
Dr. Frederick Hayes-Roth, Teknowledge Inc.

NETWORKS, LOCAL & OPEN SYSTEMS

FDDI—Fiber Distributed Data Interface, a
New High Speed LAN Standard.
Debate, Correct Interface to File Servers
Local Area Networks
ISO—OSI Pros & Cons

AI, ARCHITECTURE & EXPERT SYSTEMS

The Xenologic X1
Expert System Development Environments
Intelligent Systems for Management Decisions
AI for Manufacturing: CAD/CAM, Prod. Eng.
Knowledge Net
Expert System, Tools and Applications
Neural Based Architectures

SUPERCOMPUTERS AND SUPERMINIS

Supercomputers: ETA 10, IBM 3090, Elxsi.
Warp: Systolic Supercomputer
Second Generation Minisupers

SECURITY, TRUSTED NETWORKS

Trusted Systems
Trusted Networks
Black Magic in Computer Security

SOFTWARE, LANGUAGES & SYSTEMS

LISP Benchmarks
Future Computing Environments
COBOL 85—The New Productivity Tool
Distributed Operating Systems
Technical Directions of UNIX
Multimedia Information Environments
High Performance Transaction Systems
Management of Change In High Availability
Systems

SOFTWARE, TECHNOLOGIES

Object Oriented Model
Software Reuse—The State of the Malpractice
What's Up with Ada Compiler Technology?
Reusable Software Component Libraries
Window Systems, Diverging Views of Future

INTERESTING COMPUTER ORGANIZATIONS

AT&T Bell Labs C Machine
Alternative Computer Architectures
VAX 8800 System
Audience Open Mike, Your Provocative Proposals

OPTICS, COMPUTING & STORAGE

Optical Disks, Present and Future
New Directions in Optical Computing

CAD, PARALLEL PROCESSING

Parallel Processing for CAD Applications

DIGITAL SIGNAL PROCESSING

Digital Signal Processing

TUTORIALS: Monday, Feb. 23, 1987

"Local Networks—Changing Directions."
Harvey Freeman
"Secure Operating Systems." Lyle (Terry) Cox
"AI Machines." David Shaw
"Software Reuse: State of the Practice." Will Tracz

TUTORIALS: Friday, Feb. 27, 1987

"Reusable Software Engineering." Peter Freeman
"Practical Computer Security & Cryptography"
Russell Brand & Gilles Brassard
"Managing Knowledge Systems Software
Development." Avron Barr

REGISTER NOW AND SAVE MONEY!

TUTORIAL 1: Changing Directions in Local Networks

Instructor: Dr. Harvey A. Freeman, Vice President for Engineering, Architecture Technology Corporation, involved in all aspects of consulting, research, and publishing regarding local networks, office automation and communications systems. Co-author of Data Base Computers, co-editor of periodicals. Holds a PhD in EE from University of Illinois.

Audience: Managers, engineers, programmers, systems analysts, designers, and students. Some experience in computing and some exposure to data communications is helpful.

Abstract: This tutorial describes local area networks, a slowly maturing but still a fast growing technology in the systems market. It covers important local networking technologies, specific equipment, and companies' changing strategies.

Introduction: Overview of the field, covers early systems and their shortcomings, ISO reference model.

Ethernet vs. Token-rings: Technology, cable systems, vendors, major local networks.

Operational Issues: Servers, software, testing, strategy.

Future Directions: Standards, competition, future products, trends.

TUTORIAL 2: Secure Operating Systems

Instructors: Dr. Lyle A. Cox is the Senior engineering manager for information transport systems development for DEC. He was director of technology applications for the National Security Council at the White House, and a former professor of Computer Science at the Naval Post Graduate School. He has worked with the Departments of Defense and Energy in the areas of information systems security for the last 15 years.

Mr. Steven Lipner is senior engineering manager for servo systems development with DEC. He has been responsible for digital research and advanced development efforts in systems security since 1981. Prior to DEC, he was involved with computer security at Mitre since 1970. He participated in systems security penetration tests, development of the prototype security kernel for the PDP-11/45, and design of security enhancements to the Multics Operating System.

Audience: Computer system managers, software designers, and those concerned with operating system security.

Abstract: (1) Fundamentals of secure operating systems and real technology, (2) The employment of systems, how to keep them secure, and (3) Policy and prevention: how to encourage honesty.

TUTORIAL 3: AI Machines

Instructor: Dr. David Elliott Shaw, Associate Professor of Computer Science, Columbia University and Director of the NON-VON Project for the investigation of massively parallel

machine intelligence for AI. Doctorate from Stanford and research at the Stanford Artificial Intelligence Laboratory exploring fifth generations AI machines.

Audience: Technical and management people involved or interested in either the development or use of AI systems with a high-level perspective on the range of hardware options now available or likely to become available in the near future for AI applications.

Introduction: History of AI application-oriented hardware, projecting to future.

Significant Activities: Japanese Fifth Generation Project, DARPA's Strategic Computing Program, and related programs in Western European and Soviet block countries.

Taxonomy: Fundamental classes of machine architectures for AI.

Current AI Machines: Personal LISP machines and VLSI LISP chips, pipelined devices, vision and speech understanding.

Next Generation Architectures: Millions of processing elements, and projecting a future including non-von Neumann architectures and AI supercomputers.

TUTORIAL 4: Software Reuse: The State of the Practice

Instructor: Will Tracz, Advisory Programmer, IBM Federal Systems Division, Owego NY, and currently IBM Resident Study Fellow, Stanford University. Was responsible for the design and development of microcode support software for military and aerospace processors. He is newsletter editor of ACM SIGMICRO/IEEE TC-Micro. He has an MS in Computer Science from the Pennsylvania State University, an MS in Computer Engineering from Syracuse University, and is currently completing his PhD at Stanford University. Has taught at Syracuse University and Rochester Institute of Technology.

Audience: Those with an interest in new or advanced ideas on developing software systems to enhance programmer productivity and product quality by reusing programs, program segments, designs, or other software artifacts. Special emphasis on Ada. Knowledge of Ada is useful but not necessary.

History: Issues, motivators, inhibitors, technical, political, psychological, social and economic issues related to software reuse.

Current Practice: Emphasis on (1) developing reusable artifacts, (2) how to organize and locate reusable artifacts, and (3) how to develop artifacts. Detailed look at Japanese software factories, and survey of over 30 aerospace corporations, software factories, universities and research consortiums are surveyed. Methodology, management philosophy, and results/status. Summaries of guidelines for developing reusable software in Ada as stated by SofTech, EVB, and Honeywell.

8:30 TUESDAY, FEBRUARY 24

Welcome and Computer Society Awards

9:30 Plenary Talk

"Future Directions for High-End Workstations"

Dr. Andries Van Dam—Brown University

Supermini computer workstations developed for CAD will influence other disciplines. This talk covers the future expectations for workstation technology and applications.

10:30-12:00 TUESDAY, FEBRUARY 24

VAX 8800 System

Dileep Bhandarkar

VAX 8800 System Overview—D. McInnis: DEC

VAX 8800 Microarchitecture—Sudhin Mishra: DEC

VAX 8800 Cache and Memory—Jim Keller: DEC

VAX 8800 Design Tools and Methodology—Dennis Bak: DEC

Local Area Networks

Michael Faiman

Traffic Adaptive LAN Architecture—D. Priddy: University of Strathclyde

Local Area Broadcast Network Measurement—P. Amer: University of Delaware

Effects of Layered Protocols on Performance of a Packet Radio Network—B. Boesch: Stanford University

Expert System: Tools & Applications

Y. Muraoka

Use of an Advanced Expert Systems Tool for Fault Free

Analysis in Nuclear Power Plants—B. Frognier:

Expert-Easy Systems, Inc.

Expert System Tool with Fact/Model Representation

Environment on PSI—H. Kubono: ICOT

Towards an Expert System for Logic Circuits Synthesis—

A. DiStefano: Universita Di Catania

What's Up with Ada Compiler Technology

Geoff Mendal

Ada Computer Technology—M. Ganapathi and G. Mendal:

Stanford

Ada Compiler Technology—J. Goodenough: Software Engineering Institute

Observations on Ada—D. Luckham: Stanford

1:30-3:00 TUESDAY, FEBRUARY 24

Optical Disks—Present and Future

Bruce Lowenthal

Data Structure and Access Method Considerations

on WORM Devices—B. Lowenthal: Tandem Computers, Inc.

A Survey of Very Large Capacity On-Line File Systems

Using Optical Disks—Y. Kinouchi: Nippon Telegraph and Telephone

A Library Application Using Optical Disk—B. Kiser: Ohio College Library Corp.

Local Network File Servers: Which One?

Panel Session:

Paul Sherer: 3 COMM, Corp.
Peter Weinberger: AT&T Bell Laboratories
Mark Stein: Sun Microsystems

Gene McDaniel

Expert System Development Environments— Issues and Future Directions (Titles and authors to be announced.)

Mike Williams

Software Reuse—the State of the Malpractice

Will Tracz

The Design of Ada Systems from Reusable Components—G. Booch: Rational
STARS Reusability Guidelines—E. Wald: Naval Research Lab
Parameterization Pitfalls and Potentials—J. Goguen: SRI
Software Reuse: Motivators and Inhibitors—W. Tracz: IBM

3:30–5:00 TUESDAY, FEBRUARY 24

The Xenologic X1

Doug DeGroot

A Coprocessor for AI, LISP, Prolog, and Data Bases—T. Dobry: Xenologic, Inc.
Integration of the Xenologic X1 AI Coprocessor with General Purpose Computers—R. Ribler: Xenologic, Inc.
System Level Performance Using AI Coprocessors—A. Despain: Xenologic, Inc.

The Technical Directions of UNIX

Mark Compton

Mach: A New Kernel Foundation for UNIX Development—R. Rashid: Carnegie Mellon University
System V Directions—T. Barrese: AT&T
Directions of UNIX at the University of California at Berkeley—M. McKusick: UC Berkeley

Intelligent Systems for Management

Joseph Fiksel

Decision Support

Panel Session:

J. Kopcke: THORNE-EMI,
S. Holzman: Strategic Decisions Group,
A. Mosleh: Pickard Lowe & Garrick, Inc.,
R. Elmore: Syntelligence, Inc.

Reuseable Software Component Libraries

Rick Conn

Panel Session:

R. McCain: IBM,
S. Arnold: TI,
W. Frakes: AT&T,
P. Gauthier: CSELT,
E. Berard: EVB Software Engineering

8:30–10:00 WEDNESDAY, FEBRUARY 25

Supercomputers

Ken Wallgren

Engineering View of the ETA 10—J. Emery: ETA Systems
IBM 3090 System Design with Emphasis on the Vector Facility—S. Tucker: IBM
Engineering View of the ELXSI—L. Shar: ELXSI

Distributed Operating Systems

Hasan Alkhatib

Design Issues in Heterogeneous Distributed Computing Systems—A. Hac: John Hopkins Univ.
Branch and Bound Algorithms for Scheduling Communicating Tasks in a Distributed System—B. Greenblatt: Univ. of Southwestern Louisiana

AI for Manufacturing

William Mark

Changing the Nature of CAD/CAM with AI—C. Kempf: FMC Corp.
Knowledge-based Engineering for Production Planning and Control—I. Johnson: Garegie Group
Digital's Knowledge Network—F. Lynch: DEC

Microprocessor Applications in Korea

Jung Wan Cho

UNIX Computer Families—K. Rim: ETRI
Design of a Production Loom Monitoring System—M. Ko: Seoul National University
An Implementation of Ring Network Controller for Real Time Applications—K. Park: KAIST
TDX: A Distributed System Architecture for High Throughput Digital Switching—W. Yu: ETRI

10:30–12:00 WEDNESDAY, FEBRUARY 25

Second Generation Minisupers

Kenichi Miura

Vitesse Minisuper—C. Maples: Vitesse Electronics Corp.
Multiflow, Hardware and Software—J. Fisher: Multiflow Computer Inc.
The Personal Supercomputer—G. Culler: Culler Scientific

The Object Oriented Model

Mohammed Ketabchi

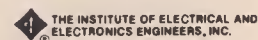
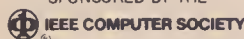
Implementation of Persistent Multi-User Object-Oriented Systems—M. Ketabchi: Univ. of Santa Clara
Object Management in HEXUS Distributed Operating System—A. Tripathi: Univ. of Minnesota
Architectural Support for Object-Oriented Programming Languages—K. Kavi: Univ. of Texas at Arlington

Trusted Networks

Stanley Ames

(Titles and author to be announced.)

SPONSORED BY THE



FOR ROOM RESERVATIONS: CATHEDRAL HILL HOTEL, San Francisco

Call (415) 776-8200 by January 30, 1987 and mention COMPCON '87 for Special Rate!

REGISTER BY FEBRUARY 14th—AND SAVE MONEY!

Please check appropriate boxes:

CONFERENCE, Tuesday-Thursday, February 24-26, 8:30 AM to 5 PM

Members: ☐ \$175 (before 2/14/87) ☐ \$220 (after 2/14/87)

Non-Members: ☐ \$230 (before 2/14/87) ☐ \$275 (after 2/14/87)

Tutorial registration fee includes lunch and one copy of the Tutorial Notes. No more than one tutorial per person per day.

TUTORIALS, Monday, February 23, 9 AM to 5 PM

Members: ☐ \$175 (before 2/14/87) ☐ \$220 (after 2/14/87)

Non-Members: ☐ \$230 (before 2/14/87) ☐ \$275 (after 2/14/87)

CHECK ONE ONLY

1. ☐ Local Networks—

Changing Directions

2. ☐ Secure Operating Systems

3. ☐ AI Machines

4. ☐ Software Reuse: State-of-the-Practice

TUTORIALS, Friday, February 27, 9 AM to 5 PM

Members: ☐ \$175 (before 2/14/87) ☐ \$220 (after 2/14/87)

Non-Members: ☐ \$230 (before 2/14/87) ☐ \$275 (after 2/14/87)

CHECK ONE ONLY

5. ☐ Reusable Software

Engineering

6. ☐ Practical System Security and Cryptology

7. ☐ Managing Knowledge

Systems Software Development

Requests for refunds must be received in writing no later than February 21, 1987. Limited tutorial attendance is on a first-paid, first-served basis. Conference registration fee includes one copy of the *Compcan Digest of Papers*, as well as admittance to the conference-hosted party.

*PAYMENT MUST BE POSTMARKED BY FEBRUARY 14 TO QUALIFY FOR ADVANCE REGISTRATION RATES. Note: AFTER 2/14/87 ALL RATES INCREASE BY \$45.

PLEASE COMPLETE THIS FORM AND RETURN TO

COMPCON Spring '87

c/o Robert M. Long, Lawrence Livermore National Laboratory
P.O. Box 808, MS L-130; Livermore, CA 94550
Telephone: (415) 422-8934.

Please make checks payable to COMPCON Spring '87.

☐ Check enclosed ☐ VISA/BankAmericard ☐ MasterCard ☐ American Express

Card No. Signature Exp. Date

Amount enclosed \$ (sum of boxes checked)

Name: First Middle Last IEEE or IEEE-CS Membership No

Organization Area Code Phone Number

Mailing Address Home Bus.

City State Zip

CM JAN 87

ON-SITE REGISTRATION will be accepted at the Cathedral Hill Hotel beginning Sunday evening, February 22, 1987.

Payment should be for sum of boxes checked. No purchase orders. Foreign checks must be in U.S. dollars drawn on U.S. bank. Credit card registration is available by phone.

Parallel Architectures and Programming**Shreekanth Thakkar**

Arrays, Non-determinism, and Parallelism: A Purely Functional Approach to Parallel Programming—P. Hudak: Yale University
 Translating Sequential Languages for Concurrent Execution—G. Graunke: Sequent Computer Systems
 Parallel Programming on Transputer Based Systems—P. Wilson: Inmos Corp.
 Experience with Three Parallel Programming Systems—S. Thakkar: Oregon Graduate Center

1:30-3:00 WEDNESDAY, FEBRUARY 25**Neural Based Architectures****Robert Dourandish**

A Neural Based Knowledge Processor—J. Voevodsky: Neuro Logic, Inc.
 Connectionists, Symbol Processing in Neural Based Architectures—D. Touretzky: Carnegie-Mellon Univ.
 Drawbacks with Neural Based Architectures—D. Partridge: New Mexico State Univ.
 Timing Dependencies in Sentence Comprehension—H. Gigly: Univ. of New Hampshire

ISO-OSI Pros and Cons**Hasan AlKathib****Panel Session:**

J. Foley: EDS Corp.,
 S. Dillon: GM Research Lab,
 D. Cheriton: Stanford Univ.,
 R. Rashid: CMU

Parallel Processing for CAD Applications**Kit Tham**

NDOT2P: A High-Performance Architectural Simulation Tool—R. Smith: MCC
 Distributed Circuit Simulation with EMU: A Case Study of Three Parallelization Strategies—S. Lucco: AT&T Bell Labs.
 The Connection Machine—General Purpose Accelerator for VLSI CAD—R. Friebrick: Thinking Machines, Inc.
 Switch-Level Simulation on a Hypercube MIMD Computer—K. Tham: Mentor Graphics, Corp.

Digital Signal Processing**Edward Lee**

Application-Specific CAD of High-Throughput Digital Filters—P. Cappello: UCSB
 Distributed Computer Architecture for Digital Signal Processing—Y. Oyang: Stanford Univ.
 Synchronous Data Flow: Describing Signal Processing Algorithms of Parallel Computation—E. Lee: UCB

3:30-5:00 WEDNESDAY, FEBRUARY 25**Plenary Talk—"Trends in Knowledge Processing: From Expert Systems to Intelligent Systems Engineering"**

Dr. Frederick Hayes-Roth—Chief Scientist, Teknowledge, Inc.
 The need to integrate knowledge processing with conventional data processing has resulted in a new class of systems, "Intelligent Systems." This talk covers R & D techniques and proto-products to aid the rapid creation of such systems.

8:30-10:00 THURSDAY, FEBRUARY 26**The AT&T Bell Labs C Machine****David Patterson**

Introduction to the CRISP Microprocessor—D. Ditzel: AT&T Bell Labs
 Architectural Innovations in the CRISP Microprocessor—D. Ditzel: AT&T Bell Labs
 Compiling Code for the CRISP Microprocessor—S. Bandyopadhyay: AT&T Bell Labs

Future Computing Environments**Ephraim Glinert**

Intelligent Assistance Without Artificial Intelligence—G. Kaiser: Columbia Univ.
 An Editor for Program Design—M. Zelkowitz: Univ. of Maryland
 STILE: A Graphical Design and Development Environment—M. Stovsky: Ohio State Univ.
 Trends in Interactive Preparation of Documents—R. Furuta: Univ. of Maryland

High Performance Transaction Systems**Andreas Reuter**

Bildschirmtext—Usage Characteristics and Performance of the German Videotext System—H. Fromm: IBM
 TPF—Capacity, Availability, and Performance—T. Scrutchin: Boole and Babbage
 The Architecture of a Stock Market System—H. Sammer: Tandem Computers, Inc.

Black Magic in Computer Security**Russell Brand**

Apparently Impossible Information Exchanges and How to Accomplish Them—G. Brassard
 Impossibly Correct Systems and How to Guarantee Them—B. Baldwin
 Perfect Authentication Passwords, ID Cards, or Extra Sensors—J. Garcia

10:30-12:00 THURSDAY, FEBRUARY 26**WARP: A Systolic Supercomputer****Thomas Gross**

The Architecture of a Systolic Supercomputer—M. Annaratone: Carnegie-Mellon Univ.
 Software Development for a Systolic Supercomputer—T. Gross: Carnegie-Mellon Univ.
 Putting Applications on a Systolic Supercomputer—J. Webb: Carnegie-Mellon Univ.

Window System Futures**Keith Lantz****Panel Session:**

J. Gosling: Sun Microsystems,
 E. Smith: CMU, E. Schoen: Stanford Univ.,
 E. Schoen: Stanford Univ.,
 R. Scheifler: MIT

Management of Change in High Availability Systems**Bob Good**

Managing Change in a High Dollar Electronic Funds Transfer Environment—B. Weintraub: Precision Business Systems
 IMS Change Management at Bank of America—D. Hughes: Bank of America
 Change Management for a Railroad Operations Control System—W. Zigmat: Southern Pacific
 Change Management in Real Time Systems—S. Beard: American Airlines

Trusted Systems**William Chase**

(Titles and authors to be announced.)

1:30-3:00 THURSDAY, FEBRUARY 26**Alternative Computer Architectures****Roger Anderson**

Black Algorithms for High Speed Computing—B. Friedlander: SAXPY Computer Corp. Inc.
 Choop—L. Higbie: Sullivan Computer Corp.
 Connection Machine—G. Rancourt: Thinking Machine Corp.

Multimedia Information Environments**Theodore Laliotis**

Multimedia Aspects of CD-I—M. Edmead: MTE Associates
 A Multimedia Form Query Language—D. Dadal: HP Labs
 Authoring System Challenges for Multimedia Environments—J. Huston: The Engineering Department

COBOL 85—The New Productivity Tool**Donald Nelson**

Productivity Features and the Future—D. Nelson: Tandem Computers, Inc.
 The Standardization of COBOL 85—J. Bantaja: Microtec Research
 A Major COBOL 85 Application—G. Hall: Eyring Research

FDDI—Fiber Distributed Data Interface**Floyd Ross**

FDDI—an Overview—F. Ross: Sperry
 FDDI System Performance—L. Green: Advanced Computer Communications
 LED Source and Fiber Specifications Issues for the FDDI Standard—D. Hanson: HP

3:30-5:00 THURSDAY, FEBRUARY 26**LISP Benchmark****Raymond Lim**

LISP Machine Architecture Issues—R. Lim: NASA Ames Research Center
 High-Level Language LISP Processor—S. Krueger: TI
 Kyoto Common LISP—F. Giunchiglia: IBUKI, Inc.

New Directions in Optical Computing**Lee. Giles**

Optical Neural Networks—D. Psaltis: California Institute of Technology
 Computational Issues in Optical Digital Computing—B. Jenkins: USC
 Optical SIMD Arrays—T. Drabik and S. Lee: UCSD

Attendees' Open Mike**Jlm Warren**

Air your ideas! You have up to ten minutes to present proposals, discuss mistakes, suggest projects, report on work in progress and communicate cautions. Sign up with the chairman to air your hotly-held heresies.

TUTORIALS

Friday 2/27/87 ALL DAY

TUTORIAL 5: Reusable Software Engineering

Instructor: Peter Freeman, Associate Professor of Information and Computer Science at the University of California at Irvine. Analysis design, and construction of advanced computer applications and training of software engineers since 1961. Author of Software Systems Principles, joint editor of two other books, author of numerous technical papers. Doctorate in Computer Science from Carnegie-Mellon University, 1970.

Audience: Technical managers and development personnel at all levels who are interested in the effective reuse of software engineering workproducts (code, designs, specifications, tc.), and the relative importance of different approaches.

Fundamentals: Basic concepts, technical issues, management issues, economics.

Current Practice: Language approaches, libraries/catalogs, component factories, software factories, portability.

Getting Started: How to set up a library, obtain programmer support and management support, make improvements.

Other Applications and Issues: Application generators, handbooks, standards.

Research: Transformational approaches, domain-modelling paradigm, design representations.

TUTORIAL 6: Practical System Security and Cryptography

Instructors: Russell Brand, member of the Computation Research Group, Lawrence Livermore National Lab. Doing graduate studies at the University of California at Berkeley, undergraduate degree from MIT. Gilles Brassard, Associate Professor of Computer Science, Université de Montreal. He has taught algorithms and cryptology at the University of California at Berkeley and does research in probabilistic algorithms, fundamental cryptography and cryptographic protocol design. Doctorate in Computer Science at Cornell University under the direction of John Hopcroft.

Audience: Those interested in practical issues of computer security, defenses against often overlooked common problems

and attacks, simple methods to plug large classes of security problems with examples from UNIX and analogies in VMS, and practical issues in computer cryptography emphasizing current computation limitations, and affordable levels of protection.

Introduction: Threats, attacks, solutions.

Log-in: Pre-login, post-login, guessing passwords, pre-built bugs, trusted hosts.

Attack Levels: Crackers, resource thieves, spies.

Solutions: Prevention, passwords, trusted hosts, detection, and post-login solutions.

Special Threats: Financial, academic, and real-time control systems.

Other: Unfriendly requests, communicating problems, VMS and UNIX source code issues, impersonations.

TUTORIAL 7: Managing Knowledge Systems Development

Instructor: Avron Barr, Aldo Ventures, offers services to AI product developers and knowledge system builders. Studied at Cornell University, at the University of California at Berkeley, and at Stanford University, and has degrees in Mathematics, Physics, and Computer Science. A Research Associate at Stanford, he was in charge of managing the large Handbook of Artificial Intelligence, a three-volume book for which he became co-editor. Barr was a co-founder of Teknowledge, Inc., a leading Knowledge Engineering firm.

Audience: Software engineers and managers interested in Knowledge Engineering, the wide utility of a new Software Engineering methodology, and knowledge system development do's and don'ts as gleaned from experience.

Fundamentals: Knowledge-based software and review of history of some typical, fielded commercial applications of the technology.

Project Manager: Learning the alternatives, hurdles, dangers, and making decisions at each stage.

Software Project Stages: Choosing application, planning, resource allocations and team skills, acquiring special-purpose knowledge engineering tools from vendors, preliminary knowledge acquisition and rapid prototyping, iterative refinement, knowledge base design as software design, systems integration and fielding, and maintenance and knowledge updating.

Future: Prediction of expected developments in the Knowledge Engineering industry with implications for computing in general.

Members of COMPCON Spring '87 Committee

General Chairman
George Michael
Lawrence Livermore National Lab.

Program Chairman
K. G. Stevens, Jr.
NASA/Ames Research Center

Program Committee:
Hasan Alkhatib
University of Santa Clara
Dennis Allison
Stanford University
Alan Bell
Xerox PARC
Frederick W. Clegg
Hewlett-Packard Company

Michael Fehling
Rockwell Science Center
Bob Fink
UC Lawrence Berkeley Lab.

Jim Gray
Tandem
Theodore A. Laliotis
Hewlett-Packard Company
Glen G. Langdon, Jr.
IBM Almaden Research Center
Creve Maples
Vitesse Electronics Corp.
Stan Mazor
Silicon Compilers Inc.

Kenichi Miura
Fujitsu America, Inc.
Yale Patt
University of California Berkeley
Allan Schiffman
Schlumberger Palo Alto Research
Alan Smith
University of California Berkeley
Will Tracz
Stanford University
Jim Warren
Publisher

Members of COMPCON Standing Committee

Sidney Fernbach
Consultant
Steering Committee Chairman
Rex Rice
Consultant
Vice Chairman, Steering Committee
Hasan Alkhatib
University of Santa Clara
Digest
Doug Coffland
Lawrence Livermore National Lab.
Computer Manager

Ramon Diaz Culinet
Audio Visual
Joseph P. Fernandez
IBM
Tutorials
Bob Fink
UC Lawrence Berkeley Lab.
Local Arrangements
Lori Goerz
Lawrence Livermore National Lab.
Administration
Dave Hunt
Lawrence Livermore National Lab.
Treasurer

Theodore A. Laliotis
Hewlett-Packard Co.
Program Support
Glen Langdon
IBM
Publicity
Roy Lee
Sandia National Lab.
Tutorials
Bob Long
Lawrence Livermore National Lab.
Registration

FOR SPECIAL COMPCON HOTEL RATES REGISTER BY: Jan. 30, 1987

LETTERS TO THE EDITOR

SDI should not be designed

To the editor:

Computer magazine is to be commended for publishing one of the more fair-minded articles concerning SDI software ("Can software for the Strategic Defense Initiative ever be error-free?," Nov. 1986). However, I still came away feeling more strongly than ever that such a system should not be designed. Let me explain why.

The main conclusion of the article can be stated as follows: If a large number of highly talented professionals spend tens of thousands of hours (or more) designing a set of programs that may be as long as several million lines, then it is highly likely that the number of errors present in these programs can be expected to decrease over time. The more time spent analyzing and testing the code, the smaller the error count becomes.

A major danger is that should such an effort ever take place, we may eventually become convinced that we can and indeed should rely on such a system as the Strategic Defense Initiative is intended to promote. We may believe that we reduced an estimated error count to a value that gives us the confidence (i.e., read arrogance) that through an exhaustive scientific effort, we have truly rendered nuclear weapons "impotent and obsolete." Of course the Soviets are already concerned that it is their weapons that may be so rendered. They are worried that US missiles and other weapons will then be available for attack or intimidation. That such an SDI system could also be used as an offensive weapon is not publicly discussed.

The existence of nuclear weapons on earth is not a problem with solely a technological solution. There is no instructor's manual containing the answer to the question, "How can the 100+ nations of the earth learn to coexist without the larger ones threatening to exterminate one another and the smaller ones striving to attain such a capability?" President Reagan and others are to be lauded for trying to find a way to rid the planet of the nuclear menace. But attempting this with a purely technological approach dramatically oversimplifies the nature of the problem. (It should be noted that the sanest technological decision the US could make at present would be to stop all nuclear weapons testing.

Not a single line of code need be written, much less tested, to do this.)

Rather than build SDI, we should be developing technologies that enhance the mutual security of all nations. Using an array of sophisticated satellites to monitor various troubled regions of the planet and making the gathered data widely available would be a start. In addition, we (and the Soviets) should be seriously developing renewable energy sources. Relying on fossil fuels is certain to lead to future wars as supplies become threatened. Finally, we should consider the suggestion of Carl Sagan (and others) to join with the Soviets in a spaceflight to Mars. Such a venture would surely be a wiser use of advanced computer technology than SDI.

There ARE valid reasons why the US and the USSR both feel the need to prepare to exterminate the other nation. But if we are ever to see a world without nuclear weapons, we shall have to begin to understand the historical, cultural, psychological, and spiritual elements that have led us to our present perilous state. Continuing to seek simplistic SDI-type solutions only perpetuates the arms race and further impoverishes both our nations.

Richard Steinberger
Senior Software Engineer
SRI International
Menlo Park, Calif.

Response to "LisaLearning" article

To the editor:

While I do not question the integrity of the authors, I was surprised to discover, in the November issue, an article ("LisaLearning") authored by two IBM employees that reviewed the user interface of an Apple Computer, Inc., product.

I personally know several computer-illiterate individuals who love the Macintosh mouse and desktop user interface metaphor. Perhaps next month Steve Jobs should author a study of the IBM System 370 user interface.

Gary Kiliany
Vice President, Engineering
Sentient Systems Technology, Inc.

COMPUTER SOCIETY EXECUTIVE COMMITTEE

President: Roy L. Russo*

IBM T. J. Watson Research Center
Route 134, PO Box 218
Yorktown Heights, NY 10598
(914) 945-3085

President-Elect: Edward Parrish*

Vice Presidents

Education (1st VP): Michael C. Mulder*
Technical Activities (2nd VP): Kenneth R. Anderson*
Area Activities: Willis K. King
Conferences and Tutorials: James H. Aylor
Membership and Information: Merlin G. Smith
Publications: J. T. Cain
Standards: Helen M. Wood

Secretary: Duncan H. Lawrie
Treasurer: Joseph E. Urban
Past President: Martha Sloan*
Division V Director: Martha Sloan
Division VIII Director: H. Troy Nagle
Executive Director: T. Michael Elliott
*Ex officio member of the Board of Governors

BOARD OF GOVERNORS

TERM ENDING 1987

Barry W. Boehm
Paul L. Borrelli
Glen G. Langdon, Jr.
Duncan H. Lawrie
Susan L. Rosenbaum
Bruce Shriver
Harold S. Stone
Wing N. Toy
Helen M. Wood
Akihiko Yamada

TERM ENDING 1988

Mario R. Barbacci
Victor R. Basili
Lorraine M. Duval
Michael Evangelist
Allen L. Hankinson
Laurel Kaleda
Ted Lewis
Ming T. Liu
Earl E. Swartzlander, Jr.
Joseph E. Urban

PUBLICATIONS BOARD

Vice President: J. T. Cain
Dharma P. Agrawal
Vishwani D. Agrawal
Dennis R. Allison
Bill D. Carroll
Michael Evangelist
James J. Farrell III
Tse-yun Feng
Lansing Hatfield
Ronald G. Hoelzeman
Sam Horvitz
Richard C. Jaeger
Willis K. King
Duncan H. Lawrie
Jack Lipovski
Ming T. Liu
Michael C. Mulder
Theo Pavlidis
David Pessel
C. V. Ramamoorthy
Bruce D. Shriver
Steve Tanimoto

SENIOR STAFF

Executive Director: T. Michael Elliott
Computer Society of the IEEE
1730 Massachusetts Ave., NW
Washington, DC 20036-1903
(202) 371-0101

(Will furnish complete roster of committees upon request.)

Editor and Publisher: True Seaborn
Director, Computer Society Press: Chip G. Stockton
Director, Conferences: William R. Habingreither
Director, Finance: Mary Ellen Curto

NEXT BOARD OF GOVERNORS MEETING:

Cathedral Hill Hotel
San Francisco, California
February 27, 1987, 8:00 a.m.-5:00 p.m.

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

President: Henry L. Bachman
Executive Vice President: Merrill W. Buckley, Jr.
Executive Director: Eric Herz



Guest Editor's Introduction

New Computers for Artificial Intelligence Processing

Benjamin W. Wah
University of Illinois at Urbana-Champaign

New computer architectures can be used to improve efficiency in the processing of some time-consuming AI tasks, but cannot overcome the combinatorial complexity of AI processing.

This special issue of *Computer* is about recent efforts to produce hardware and software architectures that support artificial intelligence (AI) applications. The earliest computer design to support AI processing was the implementation of Lisp on the PDP-6 computer and its successors, the PDP-10 and PDP-20, all made by the Digital Equipment Corporation. The half-word instructions and the stack instructions of these machines were developed with Lisp's requirements in mind. Since then, we have been seeing a proliferation of special-purpose computers that can support symbolic and AI processing. These efforts include the implementation in hardware of primitive operations fundamental to applications in AI, the design of a collection of primitive architectures to support more complex functions, and the design of system-level architectures to support one or more languages or knowledge-representation schemes.

Characteristics of AI computations

To develop a special-purpose computer to support AI processing, the requirements of AI applications must be fully

understood. Conventional numerical algorithms are usually well analyzed, and bounds on computational performance can be established. In contrast, many AI applications are characterized by symbolic processing, nondeterministic computations, dynamic execution, large potential for parallel and distributed processing, knowledge management, and open systems.

Symbolic processing. AI applications generally process data in symbolic form. Primitive symbolic operations, such as comparison, selection, sorting, matching, logic set operations (union, intersection, and negation), contexts and partitions, transitive closure, and pattern retrieval and recognition are frequently used. At a higher level, symbolic operations on patterns such as sentences, speech, graphics, and images may be needed.

Nondeterministic computations. Many AI algorithms are *nondeterministic*, that is, it is impossible to plan with the available information the procedures that must be executed and terminated. This comes from a lack of knowledge and from an incomplete understanding of the problem, and results in the need to enumerate all possibilities exhaustively when the problem is solved.

Dynamic execution. Owing to this lack of complete knowledge and to the uncertainty of the solution process, the capabilities and features of existing data structures and functions may be defined, and new data structures and functions may be created, when the problem is actually solved. Further, the maximum size for a given structure may be so large that it is impossible to allocate the necessary memory space ahead of time. As a result, memory space may have to be dynamically allocated and deallocated when the problem is solved.

Large potential for parallel and distributed computing. In parallel processing of deterministic algorithms, a set of necessary and independent tasks must be found and processed concurrently. This class of parallelism is called *AND-parallelism*. In AI processing, the large degree of nondeterminism offers an additional source of parallel processing. Tasks at a nondeterministic decision point can be processed in parallel. This latter class is called *OR-parallelism*.

Knowledge management. Knowledge is an important component in reducing the complexity of solving a given problem: More useful knowledge means less exhaustive searching. However, many AI problems may have a very high degree of inherent complexity, hence the amount of useful knowledge may also be exceedingly large. Further, the knowledge acquired may be fuzzy, heuristic, and uncertain in nature. The management, representation, manipulation, and learning of knowledge are, therefore, important problems to be addressed.

Open systems. In many AI applications, the knowledge needed to solve the problem may be incomplete because the source of the knowledge is unknown at the time the solution is devised, or because the environment may be changing and cannot be anticipated at design time. AI systems should be designed with an open concept and allow continuous refinement and acquisition of new knowledge.

Design issues

The essential issues in designing a computer system to support a given AI application can be classified into the *representation level*, the *control level*, and the *processor level* (see Table 1).

- The representation level deals with the knowledge and methods to solve the problem, and the means to represent it.

- The control level is concerned with the detection of dependencies and parallelism in the algorithmic and program representations of the problem.

- At the processor level, the hardware and architectural components needed to evaluate the algorithmic and program representations are developed.

Many current designs start with a given language or knowledge-representation scheme; hence, the representation level is already fixed. Research has been focused on automatic methods to detect parallelism, as well as on providing hardware support for time-consuming operations. However, the representation level is an important element in the design process and dictates whether the given problem can be solved in a reasonable amount of time. At this time, little has been done in providing tools (a) to aid users in collecting and organizing knowledge or (b) to aid them in designing efficient algorithms.

Hierarchy of meta-knowledge. *Domain knowledge* refers to objects, events, and actions per se, while *meta-knowledge* includes the extent and origin of the domain knowledge of a particular object, the reliability of certain information, and the possibility that an event will occur. In other words, meta-knowledge is knowledge about domain knowledge. Meta-knowledge can be considered as existing in a hierarchy. That is, meta-knowledge is involved in deciding the appropriate domain knowledge to apply, while meta-meta-knowledge is the control knowledge about the meta-knowledge. Higher level meta-knowledge is commonsense knowledge known to humans.

The use of meta-knowledge allows one to express the partial specification of program behavior in a declarative language, hence making programs more aesthetic, simpler to build, and easier to modify. Moreover, it facilitates incremental system development; that is, one can start from a search-intensive algorithm and incrementally add control information until one obtains an algorithm that may be search-free. Lastly, many knowledge-representation schemes and program paradigms, such as logic, frame, semantic network, and object-oriented languages, can be integrated with the aid of meta-knowledge.

There are many open problems related to the use of meta-knowledge: its unambiguous specification, its consistency

Table 1. Design issues in AI architectures.

Representation level
Hierarchy of meta-knowledge
Domain-knowledge representation
AI languages and programming
Control level
Truth maintenance
Partitioning and restructuring
Synchronization
Scheduling
Processor level
Micro-level architectures
Macro-level architectures
System-level architectures

verification, the learning of new meta-knowledge, and the use of appropriate statistical metrics.

Domain-knowledge representation. *Declarative representations* specify static knowledge, while *procedural representations* specify static knowledge as well as the control information that operates on the static knowledge.

Declarative representations are *referentially transparent*; that is, the meaning of a whole can be derived solely from the meaning of its parts and is independent of their historical behavior. Declarative representations offer a higher potential for parallelism than procedural representations, but are usually associated with a large search space that may partly counteract the gains of parallel processing.

In contrast, procedural schemes allow the specification and direct interaction of facts and heuristic information, hence eliminating wasteful searching. However, they may over-specify the precedence constraints and restrict the degree of parallel processing. When one chooses the appropriate representation scheme, trade-offs must be performed as regards the amount of memory space required to store the knowledge, the time allowed for making inferences, the expected usage of the knowledge, and the underlying computer architecture and technological limitations.

AI languages and programming. Conventional imperative languages are inadequate

quate to AI programming owing to its inefficiency in symbolic and pattern processing and its unacceptable programming complexity. New AI languages feature large declarative power, symbolic processing constructs, representation of information by lists, and use of recursion as the only control mechanism. Function-, logic-, and object-oriented languages are the major programming paradigms used for AI today, and hybrids of these paradigms have been developed. These languages differ in their expressive power, their ease of implementation, their ability to specify parallelism, and their ability to include heuristic knowledge. A language-oriented AI computer will inherit all the features and limitations of the language it implements.

Truth maintenance. Many AI applications are characterized by a lack of consistent and complete knowledge at the representation level. Hence, it may be necessary to modify the existing knowledge base continually and to maintain its consistency as new knowledge is acquired. *Truth maintenance* consists of recognizing an inconsistency, modifying the state to remove the inconsistency, and verifying that all inconsistencies are detected and corrected properly. The process of removing inconsistencies may be inconsistent itself, and may introduce further inconsistencies into the knowledge base.

Partitioning and restructuring. These refer to the reorganization and decomposition of the knowledge base and the AI program to achieve more efficient processing. The issues that need to be considered are similar to those considered for conventional multiprocessing and parallel processing systems, namely, granularity, static and dynamic detection of parallelism, and restructuring. However, the methods to resolve these issues are different. Owing to the nondeterminism encountered in AI processing, AI tasks may be decomposed into a large number of smaller tasks, which will influence the design of a special-purpose computer system to support AI processing. Many of the proposed AI systems have a collection of simple processing elements to execute tasks of small granularity and another collection of more complex processing elements to execute tasks of larger granularity.

The detection of parallelism is also complicated by the nondirectionality of the modes of variables, the dynamic creation and destruction of data, and the nondeter-

minism. In many languages designed for AI processing, the input/output modes of variables and the necessary data structures are defined at run time. Static analysis, allocation, and scheduling are impossible here. Dynamic detection and scheduling do not give satisfactory performance because of their relatively high overheads for tasks of small granularity. One popular solution is to require users to supply additional information in order to allow compile-time analysis. The amount of speedup that parallel processing of nondeterministic tasks will provide is not clear, although the potential for processing these tasks in parallel is great. Without appropriately guiding the search, restructuring the program, and detecting redundant computations, much of the power of parallel processing may be directed toward fruitless searches.

Synchronization. There are two levels of synchronization: *control-level synchronization* and *data-level synchronization*.

In procedural languages, if a statement precedes another statement in the program, the implication is that this statement precedes the second statement if the two statements share common variables; that is, control-level synchronization is implicit when data-level synchronization is needed. This implicit execution order may overspecify the necessary control-level synchronization in the problem.

On the other hand, if the tasks are specified as a set over a number of declarative languages, then control-level synchronization is absent, and the set of tasks can be processed in parallel if the tasks do not share common variables.

If the tasks have common variables but are semantically independent, then they have to be processed sequentially in an arbitrary order to maintain data-level synchronization. The difficulty of specifying control-level synchronization when tasks are semantically dependent is a major problem in declarative languages, such as Prolog. For example, the decomposition of a set into two subsets in Quicksort must be performed before the subsets are sorted. Hence, the tasks for decomposition and for sorting the subsets are both semantically dependent and data dependent. To overcome this problem, programmers are provided with additional primitives, such as the input/output modes of variables in a Prolog program, to specify the necessary control-level synchronization. These primitives may have

side effects and may not be able to specify completely all control-level synchronization in all situations. These problems may have to be dealt with at run time until sufficient information is available to solve them.

In short, there is a trade-off between the expressive power of the language and the implementation difficulty in designing a special-purpose computer system to support an AI language. New languages that combine the ability of functional languages to specify parallel tasks and that of logic languages to specify nondeterminism are evolving.

Synchronization is needed when the system is *message-based*, but may not be needed in systems that are *value-based* or *marker-based*. In a value-based system, multiple values arriving at a processor simultaneously are combined into a single value, hence contention will not happen and synchronization is not necessary. Neural networks and the Boltzmann machine are examples of this class. In systems supporting marker-passing, such as NETL and the Connection Machine, markers in a set represent entities with a common property and are identified in a single broadcast, hence synchronization is not necessary.

Scheduling. *Scheduling* is the selection of ready tasks to assign to available processors. It is especially important when there is nondeterminism in the algorithm. Scheduling can be static or dynamic. *Static scheduling* is performed before the tasks are executed, while *dynamic scheduling* is carried out when the tasks are executed.

The difficulty in designing a good scheduler lies in the heuristic metrics to guide the nondeterministic search. The metrics used must be efficient and accurate. Trade-offs must be made among the dynamic overhead incurred in communicating the heuristic-guiding and pruning information, the benefits that would be gained if this information led the search in the right direction, and the granularity of tasks.

In practice, the merits of heuristic guiding are not clear, since the heuristic information may be fallible. As a result, some AI architects do not schedule nondeterministic tasks in parallel. The excessive overhead coupled with the fallibility of heuristic information also leads some designers to apply only static scheduling to AI programs.

Micro-level, macro-level, and system-level architectures. The VLSI technology

that has flourished in the past 10 years has resulted in the development of many special-purpose computers.

- *Micro-level architectures* to support AI processing consist of architectural designs that are fundamental to applications in AI.

Examples of basic computational problems that are solved efficiently in VLSI are set intersection, transitive closure, contexts and partitions, best-match recognition, recognition under transformation, sorting, string and pattern matching, dynamic programming, selection, and proximity searches. Special features in AI languages that are overhead-intensive can also be supported by hardware. Examples

of these architectures include the unification hardware, tag bits for dynamic data-type checking, and hardware stacks.

- The macro-level is an intermediate level between the micro-level and the system level. *Macro-level architectures* can be made up of a variety of micro-level architectures and can perform more complex operations. Examples include the dictionary and database machines, architectures for searching, and architectures for managing dynamic data structures (such as the garbage-collection hardware).

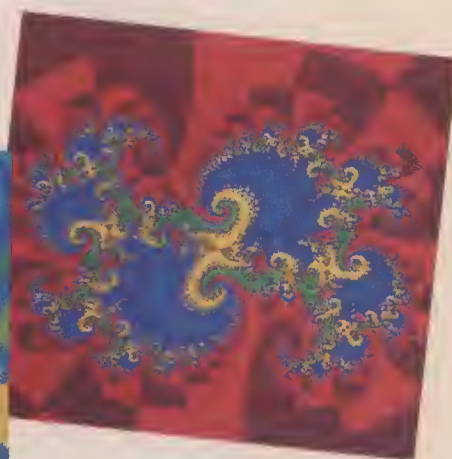
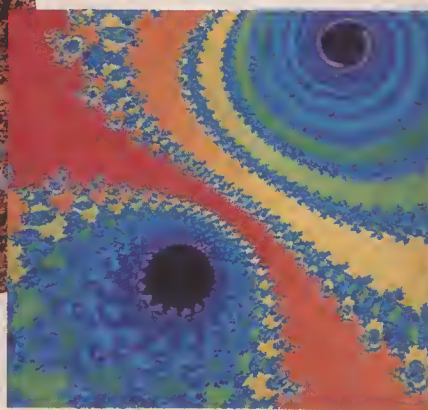
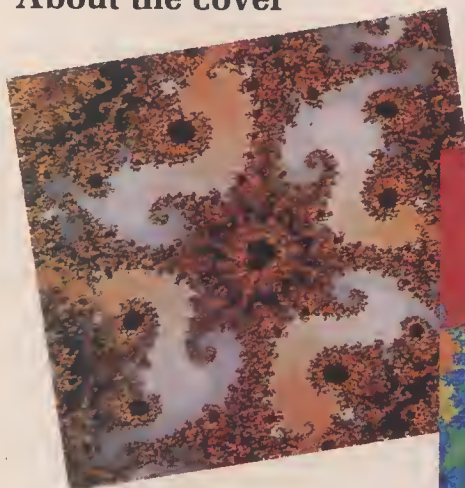
- The *system-level architectures* available today are generally oriented toward one or a few of the languages and knowledge-representation schemes and

designed to provide architectural support for overhead-intensive features in these languages and schemes. Examples include systems to support functional programming languages, logic languages, object-oriented languages, production rules, semantic networks, and special applications, such as robotics and natural-language understanding.

Design methodology

The issues classified in Table I provide a view to the design methodology for special-purpose computers that support AI processing. The various approaches can be

About the cover



The explosion of form and color here and on the cover symbolizes the complexity of AI processing—its combinatorial complexity. The enumerative nature of many algorithms used in AI applications can lead to endless searching for the correct combinations. The key to harnessing this unbridled combinatorial stampede is to establish good heuristics and efficient computers.

The graphics were created by a graphics program of an iterative process in which the output of the polynomial $y = z^2 + c$ provides the input for the same equation, whose output in turn becomes input, and so on. The images, in an infinite regress, are made up of smaller and smaller clones of the parent image. The colorless areas of the images locate a set of numbers in the complex

plane known in the field of fractal geometry as the Mandelbrot set—named after Benoit Mandelbrot of the IBM T. J. Watson Research Center.¹⁻⁴

The images were generated at the Cornell National Supercomputer Facility in conjunction with the university's Mathematics Department and Theory Center. They were computed on Floating Point System's Models 164 or 264 Array Processors attached to an IBM 3090/400 Quad Processor mainframe running VM/370 Extended Architecture with 999 million bytes of virtual address space. They were displayed on an Advanced Electronics Design model 512 graphics tube. The AED 512 was attached to the IBM 3081 via a 9600-baud RS-232 port and a Device Attachment Control Unit 7170 high-speed communications line.

They were photographed with a Canon A1 Camera equipped with a Vivitar 70-150 zoom lens with Fujicolor or Fujichrome 100 film at F9.5 and 1 second.

The photos were provided courtesy of Homer Wilson Smith of Art Matrix, Ithaca, New York, whose stock of photos of the Mandelbrot set are remarkable for their complex beauty.

1. H. W. Smith, *Mandelbrot Sets and Julia Sets*, Art Matrix Corp., PO Box 880, Ithaca, NY 14851-0880.
2. A. K. Dewdney, "Computer Recreations," *Scientific American*, August 1985, p. 16.
3. B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman and Co., New York, 1983.
4. François Robert, *Discreet Iterations*, translated by Jon Rokne, Springer-Verlag, Berlin, in press.

classified as *top-down*, *bottom-up*, and *middle-out*.

Top-down design methodology. This approach starts by defining, specifying, refining, and validating the requirements of the application; devising methods to collect the necessary knowledge and meta-knowledge; choosing an appropriate representation for the knowledge and meta-knowledge; studying problems with the given representation scheme that are related to the control of correct and efficient execution; identifying functional requirements of components; and mapping these components, subject to technological and cost constraints, into software and micro-level, macro-level, and system-level architectures.

The process is iterative. For example, the representation of knowledge and the language features may be changed or restricted if it is discovered that the functional requirements cannot be mapped into a desirable and realizable system with the given technology and within the set cost. In some projects, the requirements may be very loose and span many different applications. As a result, the languages and knowledge-representation schemes used may be oriented toward general-purpose usage. The Japanese Fifth-Generation Computer System project is an attempt to use a top-down methodology to design an integrated, user-oriented, intelligent system for a number of applications.

Bottom-up design methodology. In this approach, the designers first design the computer system; the design is based on a computational model (such as dataflow, reduction, or control-flow) and the technological and cost limitations. Both possible extensions of existing knowledge-representation schemes and languages developed for AI applications are implemented on the given system. Finally, AI applications are coded by means of the representation schemes and languages provided. This is probably the most popular approach to applying a general-purpose or existing system to AI processing. However, it may result in inefficient processing, and the available representation schemes and languages may not satisfy the application requirements.

Middle-out design methodology. This approach is a short-cut to the top-down design methodology. It starts from a proven and well-established knowledge-representation scheme or AI language (most

likely a scheme or language developed for sequential processing) and develops both the architecture and the modifications to the language or representation scheme that are necessary to adapt to the application requirements.

This is the approach taken by many designers in designing special-purpose computers for AI processing. It may be subdivided into *top-first* and *bottom-first*, although both may be iterative.

In a *top-first middle-out methodology*, studies are first performed to modify the language and representation scheme to make them more adaptable to the architecture and computational model. Primitives may be added to the language to facilitate parallel processing. Useful features from several languages may be combined. The design of the architecture follows.

In the *bottom-first middle-out methodology*, hardware support for the overhead-intensive operations enables the chosen language or representation scheme to be mapped directly into architecture and hardware. Applications are implemented by means of the language and representation scheme provided. Lisp computers are examples of computers designed with the bottom-first middle-out methodology.

The future

To support efficient processing of AI applications, research must be done in developing better AI algorithms, better AI software-management methods, and better AI architectures.

The development of better algorithms could lead to significant improvement in performance. Many AI algorithms are heuristic in nature, and upper bounds on performance to solve AI problems have not been established as they have been in traditional combinatorial problems. As a consequence, the use of better heuristic information, based on commonsense or high-level meta-knowledge and on better representation of the knowledge, could lead to far greater improvement in performance than an improved computer architecture could provide. Automatic learning methods to aid designers to acquire and manage the new knowledge in a systematic manner are very important.

Better AI software-management methods are essential in developing more efficient and reliable software for AI processing. AI systems are usually open systems and cannot be defined on the basis of a

closed-world model. The language must be able to support the acquisition of new knowledge and the validation of existing knowledge. Probabilistic reasoning and fuzzy knowledge may have to be supported. The verification of the correctness of AI software is especially difficult owing to the imprecise knowledge involved and the random way of managing knowledge in a number of declarative languages and representation schemes. Traditional software-engineering design methodologies must be extended to accommodate the characteristics of AI applications.

The role of parallel processing and innovative computer architectures lies in improving the processing time needed to solve a given AI problem. It is important to realize that parallel processing and better computer architectures cannot be used to overcome the exponential complexity of exhaustive enumeration (unless an exponential amount of hardware is used) and are not very useful in extending the solvable problem space. It is unlikely that a problem too large to be solved today by a single computer in a reasonable amount of time can be solved by parallel processing alone, even if a linear speedup can be achieved. The decision to implement a given algorithm in hardware depends on the complexity of the problem the algorithm solves and the frequency of the problem's occurrence. Problems of low complexity can be solved by sequential processing or in hardware if they are frequently encountered; problems of moderate complexity should be solved by parallel processing; and problems of high complexity should be solved by a combination of heuristics and parallel processing.

In many AI systems being developed today, tasks and operations implemented in hardware are those that are frequently executed and have polynomial complexity. These tasks and operations are identified by means of the languages or the knowledge-representation schemes supported. The architectural concepts and parallel-processing schemes applied may be either well-known conventional concepts or new concepts for nondeterministic and dynamic processing. The role of the computer architect lies in choosing a good representation, recognizing tasks for maintaining and learning meta-knowledge that are overhead-intensive, identifying primitive operations in the languages and knowledge-representation schemes, and supporting these tasks and operations in hardware and software.

In this issue

This special issue of *Computer* is a collection of articles describing a number of projects in this active area called AI computers.

The first article, "Computer Architectures for Artificial Intelligence Processing," by K. Hwang, J. Ghosh, and R. Chowkwanyun, is a survey of computers for AI processing. Existing efforts are classified as multiprocessors supporting MIMD operations, multicomputers supporting multiple SISD processing, and multipurpose computers operating in an SIMD, or multiple-SIMD, or MIMD fashion. The architecture, languages, execution paradigms, and principal applications of various AI computers are summarized.

The second article, "Software Development Support for AI Programs," by C. V. Ramamoorthy, S. Shekhar, and V. Garg, presents the problems faced in designing the software-development environment so that it will support all phases of the software-development cycle of an AI program: requirement specification, design, implementation, testing, and maintenance. The evolution of support for development of AI programming is described with respect to discrete tools, toolboxes, life-cycle support, knowledge-based tools, and intelligent life-cycle support environments.

The third article, "Symbolics Architecture," by D. A. Moon, details the design philosophy of and trade-offs in the Symbolics Lisp computers. Three levels of the architecture—system architecture, instruction architecture, and processor architecture—are discussed.

The next two articles discuss systems for the support of object-oriented programming.

The fourth article, "The Architecture of FAIM-1," by J. M. Anderson, W. S. Coates, A. L. Davis, R. W. Hon, I. N. Robinson, S. V. Robison, and K. S. Stevens, presents the design of FAIM-1, a concurrent, general-purpose, symbolic accelerator for parallel AI symbolic computations. The OIL language supported by FAIM-1 has object-oriented, logic-programming, and procedural-programming features.

The fifth article, "What Price Smalltalk?" by D. Ungar and D. Patterson, discusses the design of a Reduced Instruction Set Computer for Smalltalk-80. The requirements of the Smalltalk-80 programming environment and the valuable lessons learned by implementing clever

ideas in hardware that does not significantly improve overall performance are presented.

The sixth article, "Initial Performance of the DADO2 Prototype," by S. J. Stolfo, presents the design trade-offs of, improvements achieved by, and measured performance of DADO2, a parallel-processing system for evaluating production rules and other almost-decomposable search problems.

The last two articles are concerned with architectures for support of knowledge-representation schemes.

The seventh article, "Applications of the Connection Machine," by D. L. Waltz, discusses the architecture and applications of the Connection Machine, a system with massive parallelism. A number of applications, including document retrieval, memory-based reasoning, and natural-language processing, are presented.

The eighth article, "Connectionist Architectures for Artificial Intelligence," by S. E. Fahlman and G. E. Hinton, presents the designs of both NETL, a marker-passing system implementing semantic networks, and value-passing systems (which the authors exemplify by the Hopfield and Boltzmann iterative networks) for constraint satisfaction.

Owing to page limitations, we were unable to include two other articles originally accepted for this special issue. One, "The Architecture of Lisp Machines" by A. R. Pleszkun and M. J. Thazhuthaveetil, enumerates the runtime requirements of a Lisp system and identifies architectural requirements that must be met for good machine performance. The solutions to these requirements in a number of Lisp machines are also discussed. The other, "Computer Architecture for the Processing of a Surrogate File to a Very Large Data/Knowledge Base," by P. B. Berra, S. M. Chung, and N. I. Hachem, shows the design and performance of a proposed back-end system to support the use of surrogate files in data/knowledge bases. The articles will appear in an upcoming issue in the near future.

Despite the large number of articles in this special issue, it was not possible to cover many major projects in this area. I realize that there are many researchers, too numerous to mention individually, who have made notable contributions to the development of this area of research, and I apologize for any inadvertent omissions. Two collections of articles^{1,2} that I have compiled also provide reference sources

for some of the published articles in this exciting area. □

Acknowledgments

I would like to thank the authors and reviewers for helping to make this special issue a reality. Without them, there would be no special issue. The editor-in-chief of *Computer*, Mike Mulder, and *Computer* Editorial Board member Franklin Kuo helped me through the formalities of publication. I am also grateful to G. J. Li for his comments and to K. Lindquist for her secretarial support. I would like to acknowledge the support of National Science Foundation Grant DMC 85-19649 for this project.

References

1. B. W. Wah and G. J. Li, *Computers for Artificial Intelligence Applications*, IEEE Computer Society Press, Washington, DC, 1986.
2. B. W. Wah and G. J. Li, "A Survey of Special-Purpose Computer Architectures for AI," *ACM SIGART Newsletter*, Number 66, Apr. 1986, pp. 28-46.



Benjamin W. Wah is an associate professor in the Dept. of Electrical and Computer Engineering and in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign.

He was on the faculty of the School of Electrical Engineering at Purdue University between 1979 and 1985.

His current research interests include parallel computer architectures, artificial intelligence, distributed databases, computer networks, and theory of algorithms.

Wah was an IEEE-CS Distinguished Visitor between 1983 and 1986. He is an editor of the *IEEE Transactions on Software Engineering* and the *Journal of Parallel and Distributed Computing*. He is the program chairman of the 1987 IEEE International Conference on Data Engineering.

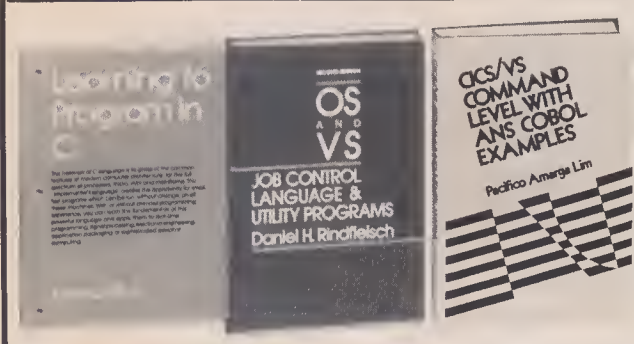
He has authored *Data Management in Distributed Systems* (University Microfilm International Research Press, 1982), and has coedited a *Tutorial on Computers for AI Applications* (IEEE-CS Press, 1986).

He received a PhD in computer science from the University of California at Berkeley in 1979.

Readers may write to Benjamin Wah about this special issue at the University of Illinois at Urbana-Champaign, Coordinated Science Laboratory, 1101 W. Springfield Ave., Urbana, IL 61801.

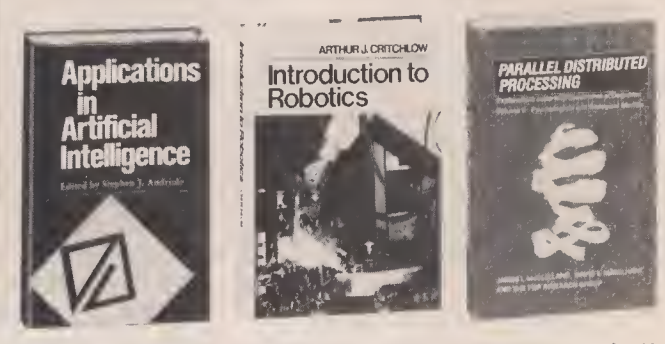
Take any 3 books for only \$1.00 each (values to \$89.90)

(Publishers' Prices shown) You simply agree to buy 3 more books—at handsome discounts—within the next 12 months



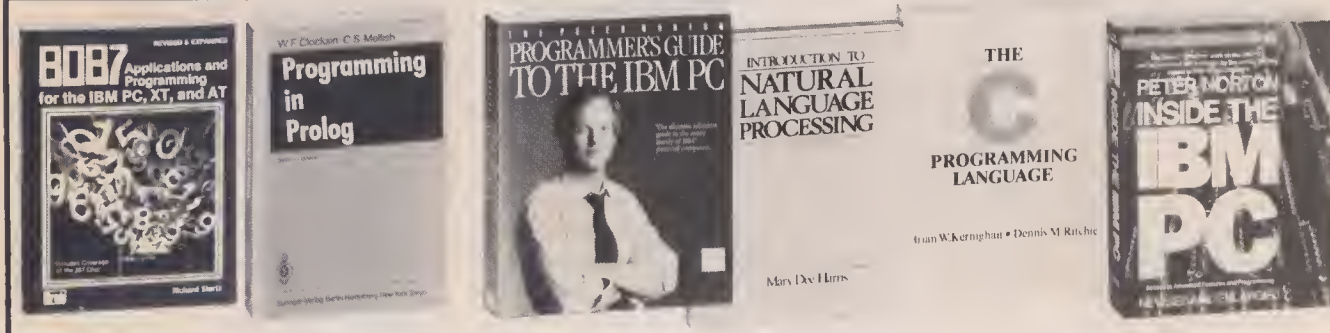
58088	\$25.00	66400	\$28.95	38230-2	\$39.95
(Softbound)				(Counts as 2 choices)	

Programming



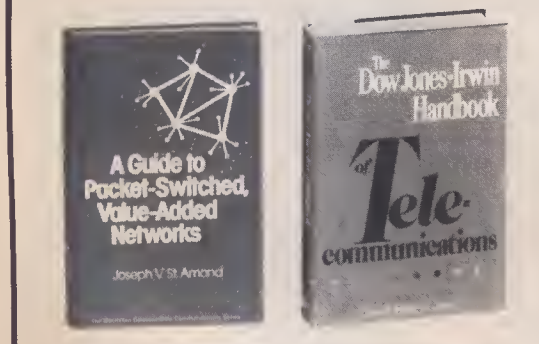
34137-2	\$49.95	56283	\$35.95	67110-2	\$45.00
(Counts as 2 choices)		(Counts as 2 choices)		(2 books count as 2 choices)	

Artificial Intelligence



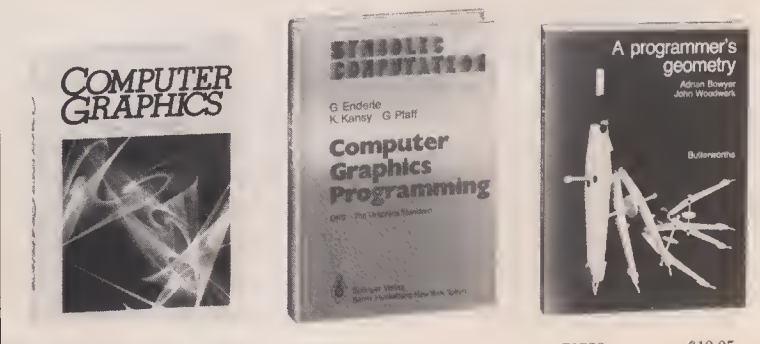
43966	\$21.95	70718	\$17.95	68430	\$19.95	56285-2	\$32.95	37206	\$24.95	55520	\$19.95
(Softbound)				(Softbound)		(Counts as 2 choices)		(Softbound)		(Softbound)	

Programming



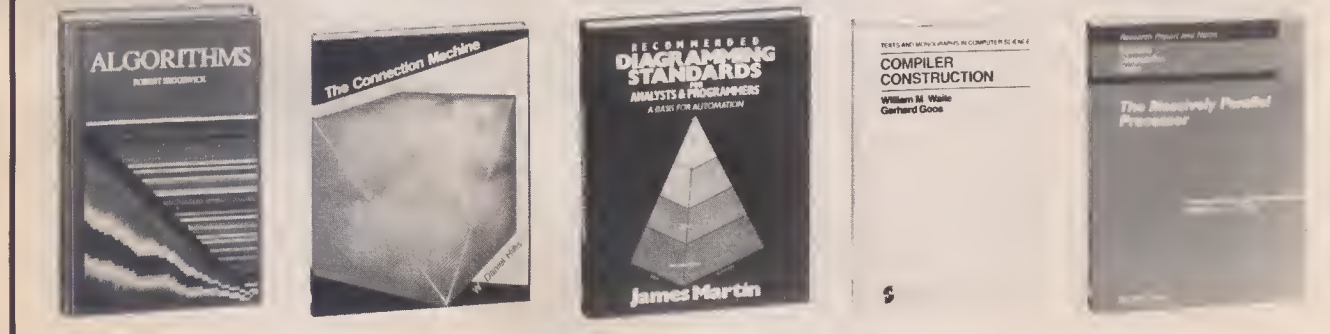
51917	\$19.95	42535-2	\$60.00
		(Counts as 2 choices)	

Data Communications



39983-2	\$39.95	39985-2	\$39.00	70722	\$19.95
(Counts as 2 choices)		(Counts as 2 choices)		(Softbound)	

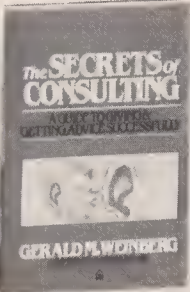
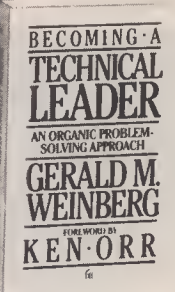
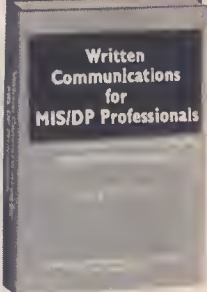
Graphics



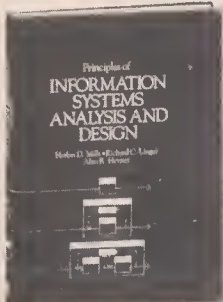
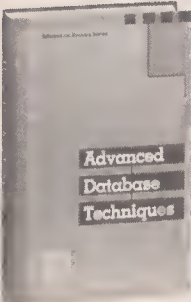
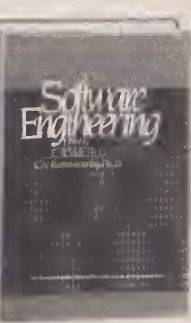
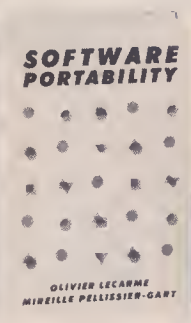
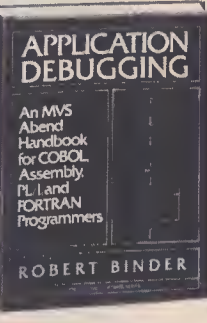
32486	\$28.95	40107	\$22.50	73646-2	\$45.00	39763	\$26.95	60650	\$30.00
				(Counts as 2 choices)					

Computer Science

The Library of Computer and Information Sciences

				
44804-3 (Counts as 3 choices)	\$87.50 77620 (Softbound)	\$25.00 35700 (Softbound)	\$28.50 37370 (Softbound)	\$21.95 88018 \$22.95

Reference

					
70119-2 (Counts as 2 choices)	\$37.50 32259-2 (Counts as 2 choices)	\$35.00	\$37.50	\$29.95	\$37.95

Database Management Software Development

THE LIBRARY OF COMPUTER AND INFORMATION SCIENCES

(the oldest, largest, and most respected computer book club in the world)

Please accept my application for trial membership and send me the three volumes indicated, billing me only \$1.00 each, plus shipping and handling. I agree to purchase at least three additional Selections or Alternates over the next 12 months. Savings range up to 30% and occasionally even more. My membership is cancelable any time after I buy these three additional books. A shipping and handling charge is added to all shipments.

No-Risk Guarantee: If I am not satisfied—for any reason—I may return my introductory books within 10 days. My membership will be canceled, and I will owe nothing.

3 books for \$1.00 each:

Indicate by number the books you want.

--	--	--

A few expensive books (as noted) count as more than one choice.

Name _____

Firm _____
(If you want subscription sent to your office)

Address _____ Apt. _____

City _____

State _____ Zip _____

(Books purchased for professional purposes may be a tax-deductible expense. Offer good in Continental U.S. and Canada only. Prices slightly higher in Canada.)

THE LIBRARY OF COMPUTER AND INFORMATION SCIENCES

(the oldest, largest, and most respected computer book club in the world)

Please accept my application for trial membership and send me the three volumes indicated, billing me only \$1.00 each, plus shipping and handling. I agree to purchase at least three additional Selections or Alternates over the next 12 months. Savings range up to 30% and occasionally even more. My membership is cancelable any time after I buy these three additional books. A shipping and handling charge is added to all shipments.

No-Risk Guarantee: If I am not satisfied—for any reason—I may return my introductory books within 10 days. My membership will be canceled, and I will owe nothing.

3 books for \$1.00 each:

Indicate by number the books you want.

--	--	--

A few expensive books (as noted) count as more than one choice.

Name _____

Firm _____
(If you want subscription sent to your office)

Address _____ Apt. _____

City _____

State _____ Zip _____

(Books purchased for professional purposes may be a tax-deductible expense. Offer good in Continental U.S. and Canada only. Prices slightly higher in Canada.)

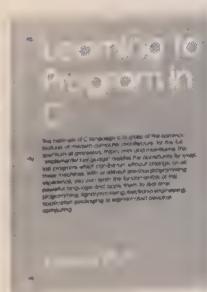
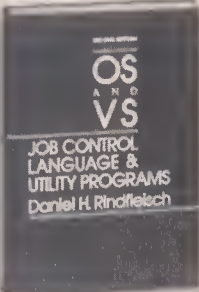
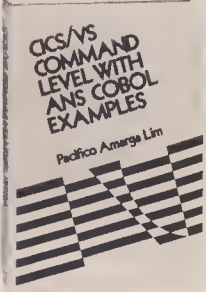
for an-
dicate
he date
to de-
of the
t want,

write
mation
8075 to
applica-

er 1/87

Take any 3 books for only \$1.00 each (values to \$89.90)

(Publishers' Prices shown.) You simply agree to buy 3 more books—at handsome discounts—within the next 12 months.

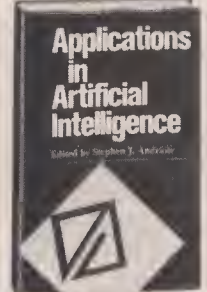
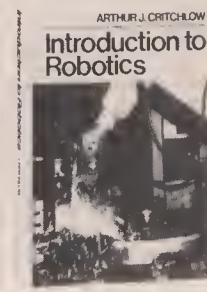
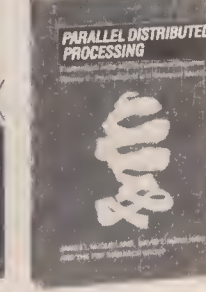




58088 \$25.00 (Softbound)

66400 \$28.95

88230-2 \$39.95 (Counts as 2 choices)

Programming

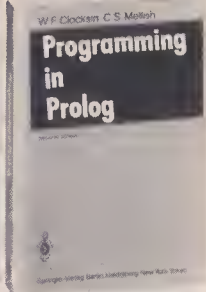
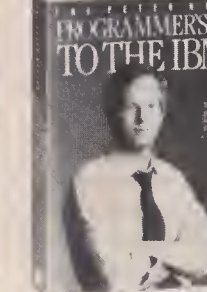
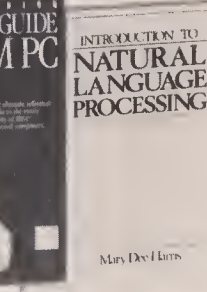
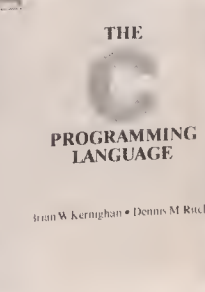
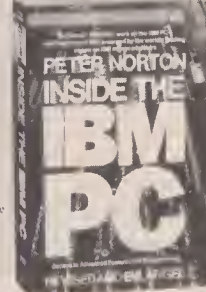




34137-2 \$49.95 (Counts as 2 choices)

56283 \$35.95 (Counts as 2 choices)

67110-2 \$45.00 (2 books count as 2 choices)

Artificial Intelligence

43966 \$21.95 (Softbound)

70718 \$17.95

68430 \$19.95 (Softbound)

56285-2 \$32.95 (Counts as 2 choices)

37206 \$24.95 (Softbound)

55520 \$19.95 (Softbound)

Programming

10-Day Free Examination

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 230 RIVERSIDE, NJ

POSTAGE WILL BE PAID BY ADDRESSEE

THE LIBRARY OF COMPUTER AND INFORMATION SCIENCES

Riverside, New Jersey 08075-9889

51

3248

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

10-Day Free Examination

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 230 RIVERSIDE, NJ

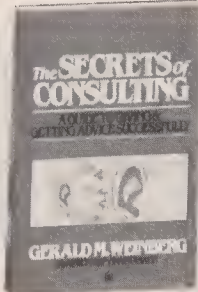
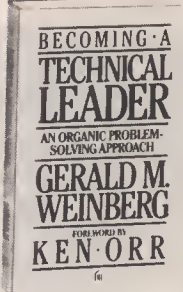
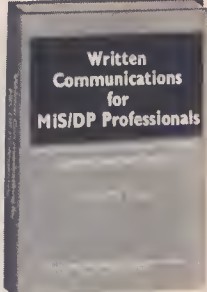
POSTAGE WILL BE PAID BY ADDRESSEE

THE LIBRARY OF COMPUTER AND INFORMATION SCIENCES

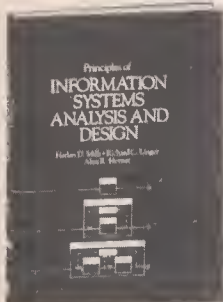
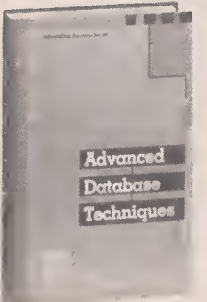
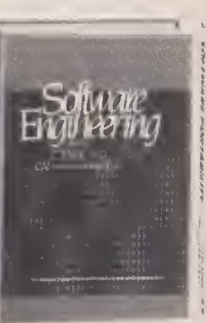
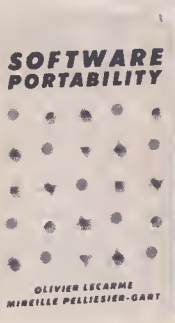
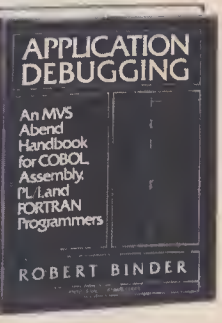
Riverside, New Jersey 08075-9889

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

The Library of Computer and Information Sciences

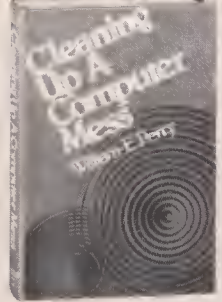
				
44904-3 (Counts as 3 choices)	\$87.50 77620 (Softbound)	\$25.00 35700 (Softbound)	\$28.50 37370 (Softbound)	\$21.95 88018 \$22.95

Reference

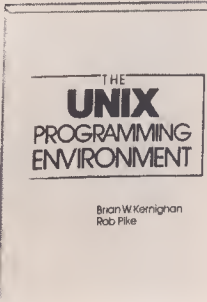
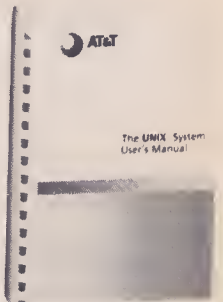
					
70119-2 (Counts as 2 choices)	\$37.50 32259-2 (Counts as 2 choices)	\$35.00	\$37.50 79168 \$29.95	52340-3 (Counts as 3 choices)	\$64.95 79164 \$29.95

Database Management

Software Development

	
69930 (Softbound)	\$28.00 38343 \$29.95

DP Management

		
74781 (Softbound)	\$21.95 85375 \$26.95	85379 (Comboud)

Operating Systems

4 Good Reasons to Join

1. The Finest Books. Of the hundreds of books submitted to us each year, only the very finest are selected and offered. Moreover, our books are always of equal quality to publishers' editions, *never* economy editions.

2. Big Savings. In addition to getting 3 books for only \$1.00 each when you join, you keep saving substantially, up to 30% and occasionally even more. (For example, your total savings as a trial member—including this introductory offer—can

easily be over 50%. That's like getting every other book free!)

3. Bonus Books. Also, you will immediately become eligible to participate in our Bonus Book Plan, with savings of up to 65% off the publishers' prices.

4. Convenient Service. At 3-4 week intervals (16 times per year), you will receive the Library of Computer and Information Sciences News, describing the Main Selection and Alternate Selections, together with a dated reply card. If you want the Main Selection, do nothing, and it will

be sent to you automatically. If you prefer another selection, or no book at all, simply indicate your choice on the card and return it by the date specified. You will have at least 10 days to decide. If, because of late mail delivery of the News, you should receive a book you do not want, we guarantee return postage.

If the reply card has been removed, please write to: **The Library of Computer and Information Sciences, Dept. 7-DU7, Riverside, N.J. 08075** to obtain membership information and an application.

 **IEEE
INTERNATIONAL
CONFERENCE ON
COMPUTER-AIDED
DESIGN**

ICCAD-87

A CONFERENCE FOR THE
ELECTRONIC CAD PROFESSIONAL

NOVEMBER 9-12, 1987
SANTA CLARA, CA

CALL FOR PAPERS

sponsored by
COMPUTER SOCIETY of the IEEE
IEEE CIRCUITS AND SYSTEMS SOCIETY
in cooperation with
IEEE ELECTRON DEVICES SOCIETY
ACM SPECIAL INTEREST GROUP
ON DESIGN AUTOMATION

THE CONFERENCE

The 1987 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN will be held NOVEMBER 9-12, 1987. The Conference is oriented towards Electrical Engineering CAD professionals, concentrating on CAD for Electronic Circuit Design.

AREAS OF INTEREST

Papers on (but not limited to) the following topics are invited:

- Simulation: Functional, Logic, Timing, Circuit, Device and Process Simulation, Modeling.
- Layout: Placement, Routing, Floor Planning, Interactive and Symbolic Layout, Circuit Extraction, Design Rule Checking.
- Test: Test Pattern Generation, Testability, Built-in-Test, Fault Simulation.
- Systems: Design Synthesis, Silicon Compilers, Expert Systems, Tool Integration, Data Base Management, Hardware Acceleration, CAD Systems.

All papers should be suitable for a 25 minute presentation, and must not have been previously published.

AUTHOR INFORMATION

Authors should submit 12 COPIES of both a 1-paragraph abstract and a more detailed description not to exceed 1500 words or six pages (double-spaced). Excessively long submissions may be returned to the authors.

Format:

The 1-PARAGRAPH ABSTRACT, typed on one separate page, should clearly and precisely state what is *new* and point out the significant results. Succinctness is required since this paragraph may be included in the Advance Program.

An author must somehow in the 1500 WORD DESCRIPTION objectively address the question why the proposed contribution is superior to prior work or what is the significance of the contribution if breaking new ground. Demonstration of superiority in algorithms and strategies with heuristics is required through a description of the programming implementation and application to "real" problems. Additional mathematical proofs are welcome. The contribution should address an area of current technical interest to the CAD professional. A clear description of the new contribution, status of the work and significant examples and results should be given. References, figures and tables are not counted as part of the 1500 words.

Submissions should include, on the first page: the title of the paper, full author names and their affiliations, complete return address and telephone number (with the individual to whom all communications should be addressed clearly identified.) In giving your return address, please consider that the communications for paper acceptance and mailing of the author kit occur in the month of July.

DATES

- Submissions should be received no later than May 1, 1987.
Send to: ICCAD-87 Secretary, Mentor Graphics Corp.,
1940 Zanker Road, San Jose, CA 95112, Tel: (408) 436-1500.
- Notification of acceptance will be JULY 1, 1987.
- Final version of Digest of Technical Papers will be due AUGUST 14, 1987.



**THE COMPUTER SOCIETY
OF THE IEEE**



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

Computer Architectures for Artificial Intelligence Processing

Kai Hwang, Joydeep Ghosh, and Raymond Chowkwanyun
University of Southern California

This article assesses a new generation of intelligent multiprocessors and multicomputers with respect to current AI technologies, advanced computer architectures, and application demands.

This article assesses different architectural approaches to the design of computers for artificial intelligence (AI) applications. We provide a classification scheme that highlights the relationship between the application domains and hardware realization of AI-oriented computer systems, as well as a comparison of AI machines and numerical computers. The emphasis is on three architectural classes:

- multiprocessors that support interactive MIMD (multiple-instruction stream and multiple-data stream) operations through a shared memory space,
- multicomputers that support multiple SISD (single-instruction stream and single-data stream) operations by means of message passing among distributed processors with local memories, and
- multipurpose computers consisting of a large number of fine-grain processor-memory nodes operating in an SIMD (single-instruction stream and multiple-data stream), a multiple SIMD, or an MIMD fashion.¹

We will examine the architecture, languages, execution paradigms, and principal applications of AI machines. This is not meant to be an exhaustive study of all reported systems. Instead, we summarize some representative systems containing unique and interesting features; these systems can provide a state-of-the-art assessment of where we stand and where we are headed. We first assess symbolic processing requirements and AI-machine charac-

teristics. Then we present an architectural taxonomy of AI machines and examine each category in some detail. (Interested readers may refer to the tutorial *Computers for Artificial Intelligence Applications*² for other relevant articles.)

Symbolic processing in AI

The acquisition, representation, and intelligent use of information and knowledge are fundamental to AI processing. An intelligent system should be able to acquire information from various external sources: visual, vocal, and written. Input from these sources is often incomplete, imprecise, or even contradictory. Symbolic processing is required for proper recognition and understanding of graphics, images, speech, and text. This involves primitive operations, such as comparison, selection, sorting, pattern matching, and logical set operations, besides arithmetic and logic operations.

Knowledge representation involves the encoding of information about objects, relations, goals, actions, and processes into data structures and procedures. The encoding should facilitate adding to, altering, and manipulating knowledge. Prominent among current approaches to knowledge representation are semantic nets, frames, scripts, procedural representation, production systems, predicate logic, and relational databases.

Processing of knowledge is required for problem solving, making logical deduc-

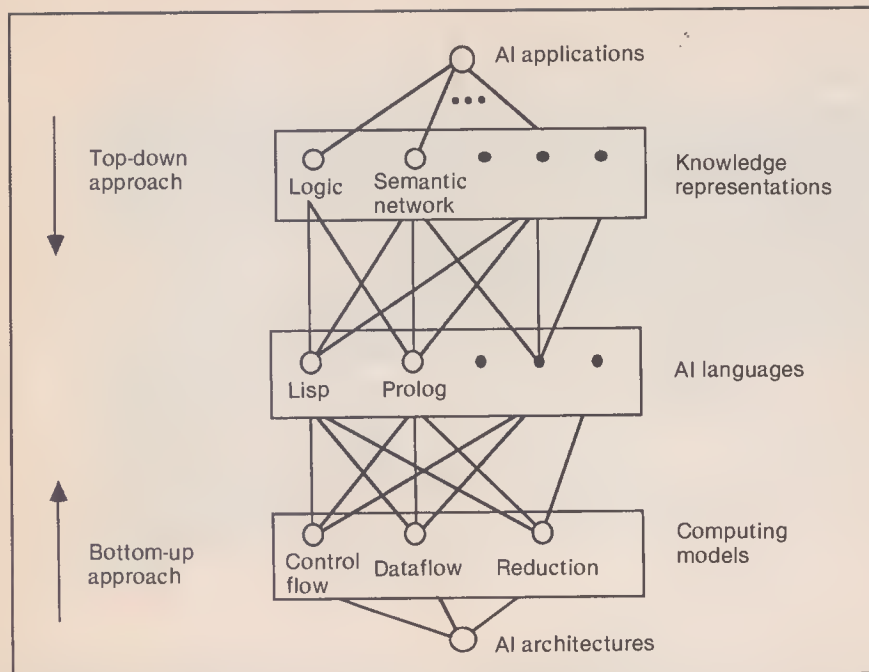


Figure 1. Two approaches to designing AI-oriented computers. (Figure courtesy of Benjamin Wah and G. J. Li.⁶)

tions, and information retrieval. The fundamental operation involved is *search*. The searching techniques include state space traversal in OR graphs, problem reduction for logic programs and AND/OR graphs, and graph search for semantic networks. The efficiency of search can be improved by heuristic guidance and pruning. Typical methods for reasoning are resolution for logic programs, forward and/or backward chaining for rule-based systems, marker propagation for semantic networks, and procedural attachments for frames. Most of these methods demand intensive memory accesses and I/O activities rather than CPU operations.

The characteristics of symbolic processing are thus fundamentally different from those of numerical processing. The main distinctions lie in symbolic knowledge representations, search-intensive operations, a large memory requirement without the locality-of-reference property, variable sizes of messages, the use of nondeterministic algorithms, interactive I/O, and knowledge database requirements.

AI-oriented computers

Owing to the radical change in the nature of the application domain, many architectural features of conventional

computers tuned for numerical applications become either inadequate or superfluous for efficient support of symbolic processing. A von Neumann architecture with centralized control presents a processor/memory bottleneck to intensive and irregular memory access patterns. Therefore, parallel and distributed processing of AI problems is desirable.³ Many AI algorithms are nondeterministic in nature, so dynamic resource allocation and load balancing are of major concern. Furthermore, the management of large knowledge bases results in a need for high I/O bandwidths and efficient interconnections among processor/memory nodes.⁴

As shown in Figure 1, the development of AI computers can be bottom-up, starting from the architectural features of the machine and working upwards to the applications they can support efficiently. For example, a computer with special hardware for data retrieval, pattern matching, and unification might be expected to support an inference system based on the logic-resolution principle. Alternatively, the design process can start with target applications and progressively work down towards the architectural basis. Key issues encompassed by either method include the development of AI programming languages, parallel algorithms, distributed operating systems and distributed problem solving, knowledge acquisition and in-

ference mechanisms, and intelligent man-machine interfaces.⁵

The choices made at each level of the hierarchy in Figure 1 are not necessarily independent and exclusive.⁶ For example, an expert system can be implemented so as to use both rule-based and resolution-based inferencing. Logic programs restricted to binary predicates can be viewed as forms of extended semantic networks, and then solved by means of graph search.⁷ Some architectures can support multiple languages and/or multiple modes of execution. Rediflow,⁸ for example, uses both reduction and macro-dataflow execution.

A taxonomy of AI machines

The taxonomy shown in Figure 2 covers AI systems that are based on knowledge and language approaches, as well as intelligent I/O interface machines. These branches are expanded in three subtrees in Figures 3-5. Key references are given after the listed AI machines in these three figures. At present, only some Lisp machines (Symbolics, LMI Lambda, TI Explorer, Xerox 1100) and a few AI-oriented machines (such as the Connection Machine) have entered the marketplace. All the others are exploratory systems under development at universities or at industrial R&D laboratories—an indication of the youth and promise of this field.

Language-based machines are designed to execute efficiently a computational model of some high-level AI language such as Lisp. They incorporate special mechanisms in hardware to implement primitive operations of the languages they support. Most systems listed in Figure 3 are either uniprocessors or have a coarse-grain architecture (that is, they consist of a small number of fairly powerful processors). The *Lisp machines* are stack-oriented and use a tagged architecture. *Prolog machines* need pattern search and unification as basic mechanisms so that they can make logical inferences based on the resolution principle. Architectures for functional languages use computational models, such as dataflow and reduction, which have asynchronous and distributed control. Therefore they can exploit a greater degree of parallelism than von Neumann computers using control-flow.

Knowledge-based AI machines (Figure 4) are governed by models for representation and manipulation of knowledge. At-

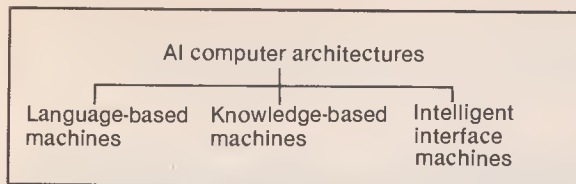


Figure 2. Three major classes of AI computer architecture.

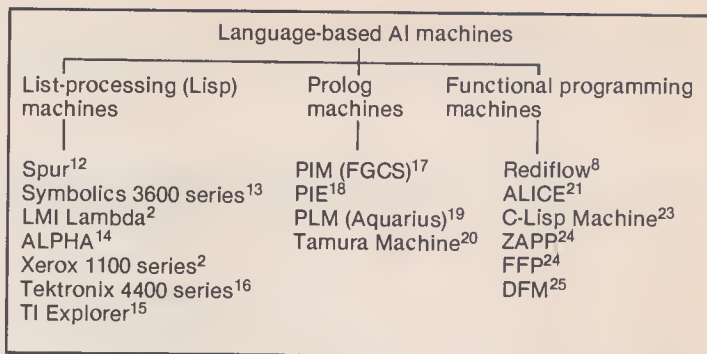


Figure 3. Language-based AI machines.

tempts at modeling the brain have inspired massively parallel computers that use thousands of processors, each having limited hardware and a small local storage capability, and functioning as an extremely reduced instruction set computer (RISC). Such connectionist architectures⁹ are based on computation models in which intelligent activity can be performed by the simultaneous execution of a large number of extremely simple operations in an MIMD/SIMD fashion.

Knowledge can also be stored as *intentional data* (rules) and *extensional data* (facts), and manipulated by rule-based systems. Object-oriented systems use *inheritance* to group together data/procedures common to a class of logically related entities. Machines for speech recognition, image understanding, and computer vision (Figure 5) provide intelligent I/O interfaces for AI systems. Below, we characterize each class of AI machines separately, and we summarize architectural features and typical applications of some representative AI systems in Tables 1-7.

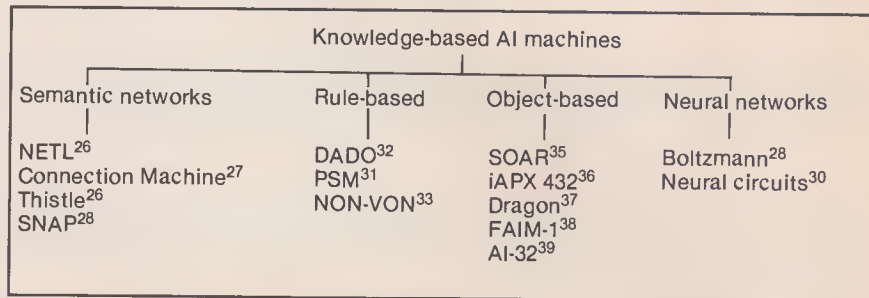


Figure 4. Knowledge-based AI machines.

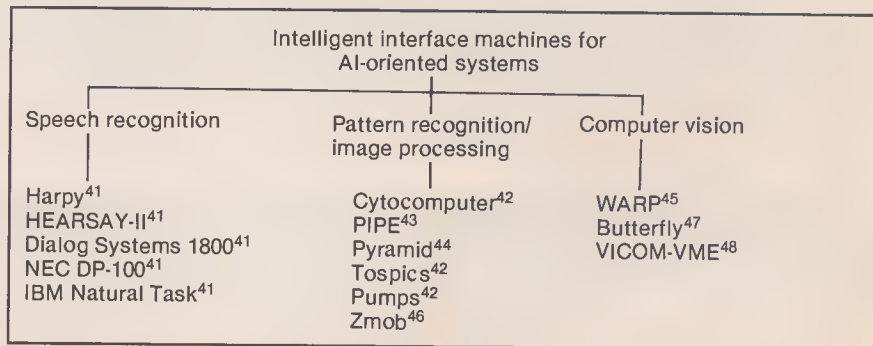


Figure 5. Intelligent interface machines.

List-processing (Lisp) machines

A Lisp program can be seen as a set of functions in which data is passed from function to function. The concurrent execution of these functions forms the basis for parallelism. The applicative and recursive nature of Lisp requires an environment that efficiently supports stack computations and function calling. The use of linked lists as the basic data structure makes an automatic storage allocation mechanism with efficient garbage collection vital. AI programs such as MACSYMA, which consists of about 230K words of compiled Lisp code, need a large memory space. Conventional computers fail to meet the above requirements. They

also do not recognize untyped data objects efficiently. Thus, fast execution of Lisp necessitates novel architectures tailored to its specific requirements.^{10,11}

As can be seen from Table I, most commercial Lisp machines¹²⁻¹⁶ have a uniprocessor architecture with tagged memory. Notable exceptions are Lambda and TI-Explorer, which are centered around the very fast NuBus and are capable of supporting more than one processor. The Xerox 1108, 1100, and 1132 are personal workstations that support the InterLisp-D system, which is designed for exploratory development of AI programs. The Symbolics 3600 shifts more operations to hardware from microcode than its ancestors. The trend of increased hardware support is also seen in the ALPHA, which offers a low process-switching time for multiple

processes through a virtual stack and an associative cache.¹⁴ Lisp machines are dealt with in more detail in an article scheduled to appear in an upcoming issue of *Computer*.¹¹

Prolog machines

Logic-oriented AI machines based on resolution directly execute programs expressed in a Prolog-like language. In the dataflow model, an intermediate dataflow language is interposed between the Prolog program and the physical hardware. A translator is required to map the Prolog programs into dataflow graphs before execution. Table 2 summarizes implementations of Prolog machines and includes instances of machines that cover both approaches.¹⁷⁻²⁰

Table 1. Commercial Lisp machines.

Manufacturer and Model	Year of Introduction	Comments
Xerox 1100 series	1981	Personal workstation supporting the Interlisp programming environment, with bytecode emulation facilities.
Symbolics 3600 series	1983	Provides direct hardware execution of data type checking at runtime, garbage collection, and instruction pipeline with prefetch.
Lisp Machines Inc. (LMI), Lambda	1983	Modularly expandable multiprocessor centered around a fast NuBus; provides almost unlimited virtual memory space.
Fujitsu ALPHA	1983	Hardware support for multiple processes through virtual stacks augmented by a cache that stores stack addresses of free variables.
Tektronix 4400 series	1984	Lower end workstations built around conventional Motorola 68010/20 processors.
Texas Instruments Explorer	1984	32-bit-NuBus-based open architecture, with a dedicated Lisp processor and multiprocessor expansion capabilities.

Table 2. Prolog machines.

Machine and Developer	System Architecture	Execution Paradigm; Parallelism used	Status
Parallel Inference Machine (PIM), ICOT, Japan.	Loosely coupled multiprocessor.	Logic (PIM-R)/Dataflow (PIM-D); AND, OR, unification parallelism.	Under construction.
Parallel Inference Engine (PIE), University of Tokyo.	Loosely coupled multiprocessor/multicomputer.	Logic; OR and unification parallelism.	Under construction.
Programmed Logic Machine (PLM), UC Berkeley.	Uniprocessor with special-purpose stacks.	Control flow; OR and unification parallelism.	Under construction.
Tamura Machine, Kobe University, Japan.	Bus-based multiprocessor with master-slave control.	Control flow; AND parallelism.	Constructed.

Table 3. AI machines for functional programming.

Machine and Developer	Architecture	Execution Paradigm Granularity	Language(s) Supported
ALICE, Imperial College, London.	Shared-memory multiprocessor connected by crossbar and two rings.	Graph reduction (dataflow at low level), fine/medium granularity.	Hope, ²² Parlog Lisp, Prolog
Rediflow, University of Utah.	Mesh-connected processor-memory nodes with dynamic load balancing.	Graph reduction/macro-dataflow, with medium to coarse granularity.	FEL ⁸
DFM, NTT, Japan.	A dataflow multiprocessor that uses packet-switched multistage network and structure memory with lazy/lenient cons mechanism.	Dataflow, with fine and medium granularity.	Valid
C-Lisp Machine, University of Kyoto, Japan.	Shared-memory multiprocessor with a master-slave control.	Stack, with medium and coarse granularity (user specified).	Concurrent Lisp
FFP Machine, University of North Carolina.	Binary tree of processors with memory at the leaf nodes.	String reduction, with fine granularity.	Formal Functional Programming Language

The execution of a Prolog-based program can be speeded up by exploiting various kinds of parallelism. *AND parallelism* refers to the simultaneous execution of logically AND-ed clauses. If two clauses have variables in common, then a successful unification requires such variables to be bound to the same set of values in both clauses. If unification for two clauses is attempted through two concurrent processes, then we have to check that the common variables have been bound consistently. This adds to an overhead that may cancel the time saved by a parallel execution of the two clauses. One solution is to group AND-ed clauses in such a way that no variable occurs in more than one group. These groups can then be executed in parallel.

Alternatively, AND-ed clauses with common variables can be chained together in producer-consumer relationships to speed up serial AND evaluation. This *stream parallelism* essentially involves macropipelining of common variables so as to obviate the need for consistency checking. The PIM-R¹⁷ implements stream parallelism by means of message blackboards that hold the common variables and lists of suspended processes.

OR parallelism refers to the concurrent search for alternate solution paths. Most logic machines favor OR parallelism, which is nondeterministic and can yield multiple solutions. Here, no overhead is incurred in consistency checking of variable bindings. However, unrestrained OR parallelism can lead to a combinatorial explosion in the number of active OR processes, and to the possibility of infinite loops.

Finally, one can implement *unification parallelism*, which refers to the parallel matching of clauses in the Prolog database with the goal clause and to the parallel instantiation of variables to constant values. The PIM and PIE¹⁸ provide hardware for parallel matching of goal clauses to clause heads in the knowledge base, and also provide for parallel instantiation of variables.

Functional programming machines

Table 3 summarizes some of the machines that have been proposed to support functional languages. Most machines use a graph or list representation for both program and data. In *demand-driven computation*, a function can be evaluated only when it is needed by another function

for which it is an argument. The *data-driven approach* evaluates a function as soon as all its arguments are available. Demand-driven evaluation requires more overhead than data-driven evaluation, but allows better control of parallelism, more selective evaluation, and a natural way of handling infinite structures. Practical approaches tend to be hybrid: ALICE^{21,22} allows low-level data-driven execution of data-intensive segments within a framework of graph reduction. Languages such as Hope²² and Parlog are compiled into a target language for direct execution on this machine.

Proposed interconnection schemes for functional programming machines include shared buses for the concurrent-Lisp processor,²³ ring for the TI-dataflow machine, tree for the FFP reduction machine, as well as the hypercube and hypertorus.²⁴ ALICE has a two-level hierarchical system in which each top-level module consists of a cluster of transputers. The tree interconnection in the FFP machine presents a bottleneck towards the root if locality does not hold, while the cluster approach has inflexible locality boundaries that decrease efficiency if structures are to be shared among clusters. The hypercube and hypertorus offer a richer interconnection scheme at increased hardware expense and more complicated dynamic routing.

Many multiprocessor architectures for functional programming support fine-grain parallelism in programs.²⁵ On such machines, nodes in the program graph represent simple processes that tend to be active for a short time and perform some simple transformation. A bus- or ring-based topology with adequate network bandwidth can provide an even distribution of workload among the processors by means of simple allocation mechanisms. However, they do not take advantage of

locality, and can rapidly degrade in performance if message traffic increases. The Rediflow reduces communications overhead by grouping several simple processes together and executing them as a sequential process. Sugimoto's machine for concurrent-Lisp requires the programmer to specify parallelism explicitly. In both ZAPP²⁴ and Rediflow, the degree of parallelism depends on the workload, since each processor has a list of local processes that can migrate to neighboring processors only if the difference in processor load is above a threshold.

Parallel processing of semantic networks

The goal of knowledge-oriented architectures is to provide efficient execution of powerful models for the representation and manipulation of knowledge. Table 4 presents some popular knowledge-representation schemes and reasoning techniques, and representative architectures associated with them. In a semantic network, nodes represent objects or concepts, and a link between two nodes signifies a specified relationship between the corresponding objects or concepts. Knowledge retrieval from a semantic net can involve searching the entire network. If the desired fact is not stored explicitly, it may have to be deduced from other stored information. Programs for knowledge retrieval often spend most of their time repeating a few basic operations, such as logic operations on sets, sorting, pattern matching against a set of assertions, and deducing facts from semantic inheritance networks. Speedup can be obtained by simultaneously initiating these operations from many nodes of the network. Ideally, each concept (node) should be allocated to a distinct pro-

Table 4. Knowledge representations and related AI machines.

Knowledge Representation	Reasoning Technique	Machine
Predicate logic	Resolution	FGCS
Data encapsulation	Inheritance	iAPX 432
Production systems	Forward chaining	DADO
Frame	Procedural attachments	Apiary
Procedures	Program execution	Zmob
Semantic network	Marker propagation	NETL
Neural networks	Energy minimization	Boltzmann

Table 5. AI machines for production systems and semantic networks.

System and Developer	Architectural Features	Languages Supported	Principal Applications
Connection Machine, Thinking Machines, Inc., Cambridge, Mass. (marketed, 1986).	An SIMD machine with 2 ¹⁶ simple processor/memory nodes interconnected as a hypercube.	Executes Connection Machine Lisp (CmLisp).	Processing of knowledge represented by semantic networks; other applications that can exploit fine-grained parallelism in traversing a search space.
Thistle, Carnegie Mellon University, Pittsburgh (conceptual machine).	Massively parallel fine-grained system that uses both value and marker passing mechanisms.	Unknown	Semantic network processing; iterative relaxation for low-level vision.
DADO2, Columbia University (fabricated, 1986).	A binary-tree structured multiprocessor of 1023 nodes operating in SIMD/MIMD/multiple-SIMD modes.	Superset of PL/M, parallel versions of C, Lisp.	Expert systems implemented as production systems; speech recognition.
SNAP, University of Southern California, Los Angeles (conceptual machine).	A 2D mesh of identical cells with content-addressable memory; combines associative and cellular array processing.	High-level instructions built upon a set of microinstruction primitives.	Semantic-net processing; production systems; and discrete relaxation for vision.

cessor and the interconnections between these processors should be flexible enough to represent the relationships between the corresponding concepts.

This ideally requires a large number of processor elements with programmable logical connections so that the topology can be configured to suit the problem. If the silicon area is limited, the processors must be simple and have little local memory, and the interconnection network should be regular. However, the memory is distributed throughout the machine and is more intelligent because it can modify itself in many places simultaneously through the local processors. The fundamental inefficiency of the von Neumann memory, in which only a few locations can change state at any given time, is thus avoided. The above considerations are reflected in proposed massively parallel, fine-grained architectures such as NETL and Thistle machines²⁶ as well as in semantic net processors actually being built, such as the Connection Machine.²⁷ The SNAP is an array-structured machine for processing semantic networks, and is specially designed for VLSI implementation.²⁸

The power of the Connection Machine stems from its ability to store information in the reconfigurable virtual interconnection patterns of the cells, and to execute concurrently the same simple operation on a very large number of cells. It is claimed that the Connection Machine is also effective for applications like object recogni-

tion and unstructured text retrieval, which are not expressed in the semantic network framework, but require lots of searching and sorting. There is a profound change in the type of algorithms that are meaningful for use with this machine. For example, one may no longer need to presort a large set of data in the order in which it is needed, since the Connection Machine can sort its entire memory in 30 msec.²⁹

Radical computer models that are patterned after neural networks are also being developed.³⁰ Table 5 outlines some AI machines for processing semantic networks²⁶⁻²⁸ and some for implementing production systems.³¹⁻³³

Rule-based systems

Expert systems like R1 and XSEL have been written as rule-based production systems in specialized languages, such as OPS5.³¹ The basic algorithm for the execution of production systems goes through a three-phase rule-firing cycle. First, the condition clauses of all rules are "matched" against the working memory, which represents the state of the world. One of the successfully matched rules is then "selected" according to some control strategy. Finally, in the "act" phase, the selected rule is executed or fired, and the working memory is updated. This is done in a serial fashion and is inherently very slow.

Among the special architectures designed to speed up the execution are the DADO³² and the Production System Machine (PSM).³¹ DADO uses a binary tree as the basic underlying interprocessor communication network. The internal tree nodes are used to store production system rules and elements of the working memory, while the root is used to select one of the successful rules and to broadcast to the entire tree the resulting changes to be made. Speedup results from storing the state of the system in a distributed fashion, and by performing matches and state updates in parallel. The binary tree topology is favored because it can be efficiently implemented in VLSI technology, and can also easily handle broadcasts from the root to a large number of recipients. Since in the execution of production systems, each node talks only with its ancestors, the binary tree bottleneck is avoided.

Object-based AI machines

Object-oriented programming³⁴ is based on the encapsulation of data, together with all the procedures that manipulate it, into a uniform type—the *object*. Objects can only interact with one another by sending messages. A representative language like Smalltalk-80 does not require type specification for variables, can link subroutines on-the-fly, and can

manage varying data structures dynamically. A reasonably fast implementation of such a language requires efficient hardware support for a high-level instruction set, dynamic data typing, frequent and expensive procedure calls, and an object-oriented storage management.³⁵

These requirements are somewhat similar to those for Lisp machines. Therefore, it is not a big surprise that features such as parallel checking of data tags, a stack-oriented execution, and duplicate "shadow" register sets for fast trapping and procedure calls are incorporated in most object-oriented machines. Since objects are created, modified, and destroyed frequently, and since they all reside in the same virtual memory, a large virtual memory space with dynamic storage allocation and efficient garbage collection is provided. An object-oriented storage also requires schemes for object protection and a mechanism for quick retrieval of objects from memory. Table 6 shows key features of object-oriented machines.³⁵⁻³⁹

The first commercially available object-oriented architecture, the Intel iAPX 432, was a complex instruction set computer (CISC)-based multiprocessor.³⁶ In contrast, the SOAR project³⁵ extended the RISC philosophy to an object-oriented programming environment by shifting the burden of managing objects and providing complicated operations to the software. The RISC approach is also taken for the Dragon, a 32-bit super-workstation being designed at Xerox Palo Alto Research Center.³⁷ This tightly coupled system uses up to 10 identical processors, which access the main memory through a common bus.

The bus interface is through fully associative instruction and data caches. A floating-point accelerator for graphics and simulation applications is incorporated in each processor.

The FAIM-1 is a multicomputer system designed to provide efficient hardware support for an object-oriented language called OIL, in which atomic objects can be expressed in a logical or procedural fashion.³⁸ A distinctive feature of this project is the transfer of many functions from software to hardware contained in specialized memory subsystems. These units relieve the processor from many of the storage management duties, which are typically very time-consuming in AI applications. Excluded from Table 6 are object-based systems that are implemented in software and run on Lisp machines, and more general-purpose machines, such as the Intel iPSC, whose memory-enhanced version is supporting MACE, a testbed for distributed AI problem solving.⁴⁰

Intelligent I/O interface machines

AI systems often provide speech analysis, natural-language understanding, image processing, and computer vision as intelligent interfaces between man and computers. While proposed schemes for natural-language understanding are implemented in software at present, computer architectures have already emerged to support speech-understanding⁴¹ and image-understanding⁴²⁻⁴⁸ applications.

Some of these systems are highlighted in Table 7. These systems are not pure symbolic processors. In fact, low-level processing of speech and images involves substantial numerical calculations; processing of symbolic representations of phonemes and images is carried out at higher levels. The Cytocomputer⁴² has an interesting pipeline architecture specially designed for biomedical image processing. The Warp⁴⁵ is based on a programmable systolic array. It is specially designed to provide computer vision for an autonomous vehicle.

Zmob⁴⁶ and Butterfly⁴⁷ are examples of systems that can perform both symbolic and numerical processing. Both of them use up to 256 identical microprocessors each for concurrent, distributed computing. As in the Connection Machine, the main cost of these two systems is in the communication networks. The Zmob connects its Z80-based processing nodes through a fast slotted-ring to achieve a total data-in-motion rate of 14G bytes per second for a configuration with 256 processors. The local memories of the Butterfly collectively form a globally shared memory space. A processor can access another processor's memory via the butterfly network. Typically, it takes about two microseconds to complete a local memory access and about six microseconds to access a remote memory by means of the network. The Butterfly machine is especially attractive for solving large-grain problems that demand general interconnects among the processors. The Connection Machine also supports general connect, but it is more tuned towards fine-grain problems.

Table 6. Object-oriented AI machines.

Name and Developer	Architecture, Language Supported	Execution Paradigm	Status
iAPX 432, Intel Corp.	Multiprocessor-expandable around shared bus; for Ada and "purer" object-oriented languages.	Both stack and three address instructions executed.	Marketed in 1981.
SOAR, UC Berkeley.	Uniprocessor for Smalltalk.	Register-based expression evaluation with dynamic type-checking using tags.	Implemented on a chip, 1984.
FAIM-1, Fairchild, Palo Alto, Calif.	Multiprocessor with hexagonal mesh topology; executes OIL.	Stack-based, with parallel tag-checking hardware.	Under construction.
Dragon, Xerox, Palo Alto, Calif.	Tightly coupled multiprocessor that uses a common bus; supports Cedar, Interlisp and Smalltalk.	Executes both stack and memory-to-memory instructions.	Under construction.
AI-32, Hitachi, Japan.	Microprogrammable uniprocessor; supports Smalltalk and others.	Stack-based with tagged operands.	Implemented on a chip, 1986.

Table 7. Intelligent interface machines for AI systems.

System and Developer	Architectural Features	Applications	Status
Harpy Machine, Carnegie Mellon University, Pittsburgh.	Multiple microprocessors accessing shared memory through common bus.	Speech recognition.	Implemented, uses five microprocessors, 1979.
Butterfly, BBN, Inc., Cambridge, Mass.	An MIMD machine consisting of up to 256 processor/memory nodes interconnected through a butterfly switching network.	Computer vision, VLSI simulation, cognitive modeling, numerical applications.	Marketed in 1985.
Warp, Carnegie Mellon University, Pittsburgh.	A linear systolic array of programmable processors.	Image processing, vision for autonomous vehicles.	10-processor system operational, 1986.
PIPE, National Bureau of Standards, Washington, DC.	Pipelined processor with identical modular stages operating in SIMD or restricted MIMD mode.	Front-end processor for low-level image processing.	Modules built, 1985.
Pyramid, University of Washington, Seattle.	A pyramid of multilayered array processors.	Image processing, spatial problem solving, graphics.	Prototype with 85 processing elements under development, 1985.

At present, most AI machines are exploratory in nature and are still in the research and development stage. Hardly any performance statistics are available for these projects. Coarse-grain parallel processors are presently used in most numerically oriented supercomputers. Fine-grain multiprocessors or multicomputers have potential for both numerical and symbolic computations. We observe a shift towards highly parallel architectures as hardware costs keep decreasing, together with a shift of functions from software to hardware, and the emergence of intelligent memories obtained by incorporating processing logic into memory units. The conventional separation of processor and memory is getting blurred. In the computer market, superworkstations and mini-supercomputers with some degree of AI capability are emerging as commercial products. To integrate both numerical and symbolic processing into a single system will be very difficult because of the conflicts in processing requirements. The AI machine designer must be aware of these conflicting goals. To guarantee higher performance, AI machines have to be treated as special-purpose computers with the architectural features discussed in this article.

The development of high-performance and highly intelligent AI machines will be evolutionary rather than revolutionary. The challenges must be met by building on solid foundations of artificial intelligence, software engineering, computer architec-

ture, VLSI technology, expert systems, speech recognition, computer vision, networking and supercomputers.⁴¹ Advances in these areas will help develop the next generation of computers. Future efforts in developing highly parallel systems for AI applications should focus on concurrent programming languages and their compilers, distributed software environments, effective knowledge representation and manipulation, fast nondeterministic search techniques, intelligent I/O interfaces, better matching between parallel hardware architectures and parallel AI algorithms, and the exploration of new technologies, such as neural computations³⁰ and optical computing. Without major breakthroughs in these areas, AI machines will be far from being perfected.⁴⁹ It may take more than one generation of humans to produce the next generation of superintelligent computers. The authors suggest that multinational cooperative efforts be launched to reduce the development overhead and to avoid unnecessary mistakes and conflict of interests. □

Acknowledgments

The authors wish to acknowledge research support from NSF Grant DMC-84-21022 and partial support from AFOSR Grant 86-0008 and ONR Contract N66001-85-D-0203.

References

1. K. Hwang and J. Ghosh, "Supercomputers and Artificial Intelligence Machines," in *Computer Architecture: Concepts and Systems*, V.M. Milutinovic, ed., Elsevier Science, New York, 1987.
2. *Computers for Artificial Intelligence Applications*, B. W. Wah and G. J. Li, ed., IEEE Computer Society Press, Washington, DC, May 1986.
3. P. C. Treleaven, "The New Generation of Computer Architecture," *Proc. Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1983, pp. 402-409.
4. C. Hewitt and H. Lieberman, "Design Issues in Parallel Architectures for Artificial Intelligence," *Proc. COMPCON Spring*, IEEE, Feb. 1984, pp. 418-423.
5. M. F. Deering, "Hardware and Software Architectures for Efficient AI," *Proc. Nat'l Conf. Artificial Intelligence*, AAAI, Aug. 1984, pp. 73-78.
6. B. Wah, lecture notes used for a presentation during the Tutorial on Computers for AI Processing, Int'l Conf. Parallel Processing, Aug. 19, 1986.
7. L. Bic, "Execution of Logic Programs on a Dataflow Architecture," *Proc. 11th Int'l Symp. Computer Architecture*, IEEE/ACM, June 1984, pp. 290-296.
8. R. M. Keller, F. C. H. Lin, and J. Tanaka, "Rediflow Multiprocessing," *Proc. COMPCON Spring*, IEEE, July 1984, pp. 410-417.
9. S. E. Fahlman and G. E. Hinton, "Connectionist Architectures for Artificial Intelligence," *Computer*, Vol. 20, No. 1, Jan. 1987 (this issue).
10. R. P. Gabriel, *Performance and Evaluation of Lisp Systems*, The MIT Press, Cambridge, Mass., 1985.
11. A. R. Pleszkun and M. J. Thazhuthaveetil, "The Architecture of Lisp Machines," *Computer* (to appear).
12. G. S. Taylor et al., "Evaluation of the SPUR Lisp Architecture," *Proc. 13th Annual Int'l*

Symp. Computer Architecture, IEEE/ACM, June 1986, pp. 444-452.

13. D. A. Moon, "Architecture of the Symbolics 3600," *Proc. 12th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1985, pp. 76-83.
14. H. Hayashi, A. Hattori, and H. Akimoto, "Alpha: A High-Performance LISP Machine Equipped with a New Stack Architecture and Garbage Collection System," *Proc. 10th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1985, pp. 154-161.
15. Texas Instruments, *TI Explorer Technical Summary*, technical brochure, Texas Instruments, Austin, Tex., 1984.
16. W. E. Suydam, "AI Becomes the Soul of the New Machines," *Computer Design*, Vol. 25, No. 2, Feb. 1986, pp. 55-70.
17. K. Murakami et al., "Research on Parallel Machine Architecture for Fifth-Generation Computer Systems," *Computer*, Vol. 18, No. 6, June 1985, pp. 76-92.
18. H. Tanaka, "A Parallel Inference Machine," *Computer*, Vol. 19, No. 5, May 1986, pp. 48-54.
19. T. P. Dobry, A. M. Despain, and Y. N. Patt, "Performance Studies of a Prolog Machine Architecture," *Proc. 12th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1985, pp. 180-190.
20. N. Tamura and Y. Kaneda, "Implementing Parallel Prolog on a Multiprocessor Machine," *Proc. Int'l Symp. Logic Programming*, IEEE, May 1984, pp. 42-48.
21. J. Darlington and M. Reeve, "ALICE and the Parallel Evaluation of Logic Programs," technical report, Imperial College of Science and Technology, London, England, June 1983.
22. D. Pountain, "Parallel Processing: A Look at the ALICE Hardware and Hope Language," *Byte*, Vol. 10, No. 5, May 1985, pp. 385-395.
23. S. Sugimoto et al., "A Multi-Microprocessor System for Concurrent LISP," *Proc. Int'l Conf. Parallel Processing*, Aug. 1983, pp. 135-143.
24. S. R. Vegdahl, "Architectures that Support Functional Programming Languages," in *Advanced Topics in Computer Architecture*, V.M. Milutinovic, ed., Elsevier Science, New York, 1987.
25. M. Amamiya et al., "Implementation and Evaluation of a List-Processing Oriented Data Flow Machine," *Proc. 13th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1986, pp. 10-19.
26. S. E. Fahlman and G. E. Hinton, "Massively Parallel Architectures for AI: NETL, THISTLE and BOLTZMANN Machines," *Proc. Nat'l Conf. Artificial Intelligence*, AAAI 1983, pp. 109-113.
27. W. D. Hillis, *The Connection Machine*, The MIT Press, Cambridge, Mass., 1985.
28. D. I. Moldovan and Y. W. Tung, "SNAP: A VLSI Architecture for Artificial Intelligence Processing," *J. Parallel and Distributed Computing*, Vol. 2, No. 2, May 1985, pp. 109-131.
29. K. A. Frenkel, "Evaluating Two Massively Parallel Machines," *Comm. ACM*, Vol. 29, No. 8, Aug. 1986, pp. 752-759.
30. J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model," *Science*, Vol. 233, No. 4764, Aug. 1986, pp. 625-633.
31. A. Gupta et al., "Parallel Algorithms and Architectures for Rule-Based Systems," *Proc. 13th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1986, pp. 28-39.
32. S. J. Stolfo and D. P. Miranker, "The DADO Production System Machine," *J. Parallel and Distributed Computing*, Vol. 3, No. 2, June 1986, pp. 269-296.
33. D. E. Shaw, "NON-VON's Applicability to Three AI Task Areas," *Proc. Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, Aug. 1985, pp. 61-70.
34. M. Stefik and D. Bobrow, "Object-Oriented Programming: Themes and Variations," *AI Magazine*, 1985, pp. 40-61.
35. D. Ungar et al., "Architecture of SOAR: Smalltalk on a RISC," *Proc. 11th Int'l Symp. Computer Architecture*, IEEE/ACM, June 1984, pp. 188-197.
36. Intel Corp., "Introduction to the iAPX 432 Architecture," manual, Order No. 171821-001, 1981.
37. L. Monier and P. Sidhu, "The Architecture of the Dragon," *Proc. COMPCON Spring*, IEEE, 1985, pp. 118-121.
38. A. L. Davis and S. V. Robison, "The Architecture of the FAIM-1 Symbolic Multiprocessing System," *Proc. Ninth International Joint Conf. Artificial Intelligence (IJCAI)*, Aug. 1985, pp. 32-38.
39. T. Nojiri, S. Kawasaki, and K. Sakoda, "Micro-programmable Processor for Object-Oriented Architecture," *Proc. 13th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1986, pp. 74-81.
40. L. Gasser, "MACE—A Multi-Agent Computing Environment," tech. report, Computer Science Dept., University of Southern California, March 1986.
41. *Next-Generation Computers*, E. A. Torrero, ed., IEEE Press, New York, 1985.
42. *Computer* (special issue on computer architectures for image processing), K. Hwang, guest ed., Vol. 16, No. 1, Jan. 1983.
43. E. W. Kent, M. O. Shneier, and R. Lumia, "PIPE (Parallel Image-Processing Engine)," *J. Parallel and Distributed Computing*, Vol. 2, No. 1, Feb. 1985, pp. 50-78.
44. S. L. Tanimoto, "A Hierarchical Cellular Logic for Pyramid Computers," *J. Parallel and Distributed Computing*, Vol. 1, No. 1, Aug. 1984, pp. 105-132.
45. M. Annaratone et al., "Warp Architecture and Implementation," *Proc. 13th Annual Int'l Symp. Computer Architecture*, IEEE/ACM, June 1986, pp. 346-356.
46. M. Weiser et al., "Status and Performance of the ZMOB Parallel Processing System," *Proc. COMPCON Spring*, IEEE, 1985, pp. 71-73.
47. W. Crowther et al., "The Butterfly Parallel Processor," *Newsletter, Computer Architecture Technical Committee*, IEEE Computer Society, Sept./Dec. 1985, pp. 18-45.
48. W. K. Pratt, "A Pipeline Architecture for Image Processing and Analysis," *Proc. Workshop for Computer Architecture for Pattern Analysis and Image Database Management*, IEEE, Nov. 1985, pp. 516-520.
49. K. Hwang, R. Chowkwanyun, and J. Ghosh, "Computer Architectures for Implementing Artificial Intelligence Systems," Tech. Report CRI-86-25, Computer Research Institute, University of Southern California, Los Angeles, Calif., Nov. 1986.



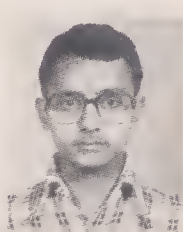
Kai Hwang, a professor of EECS at the University of Southern California, serves as the director of the Computer Research Institute at USC.

Hwang, who received his PhD from the University of California at Berkeley, has published over 70 scientific publications on computer organization, parallel algorithms, supercomputers, multiprocessors, VLSI and dataflow architecture, and AI-oriented computers.

He authored *Computer Arithmetic* (Wiley, 1979) and *Computer Architecture and Parallel Processing* (McGraw-Hill, 1984); he edited *Supercomputer Design and Applications* (IEEE-CS Press, 1984). The founding editor of the *Journal of Parallel and Distributed Computing*, published by Academic Press, he is also a series editor of the new McGraw-Hill Book Series in *Supercomputing and Artificial Intelligence*.

He has served as an IEEE-CS Distinguished Visitor and was elected an IEEE Fellow for contributions to digital arithmetic, systems architectures, and parallel processing. He chaired the IEEE/ACM 1985 Seventh Symposium on Computer Arithmetic and the 1986 International Conference on Parallel Processing.

Readers may write to Kai Hwang at the Computer Research Institute of the University of Southern California, Los Angeles, CA 90007.

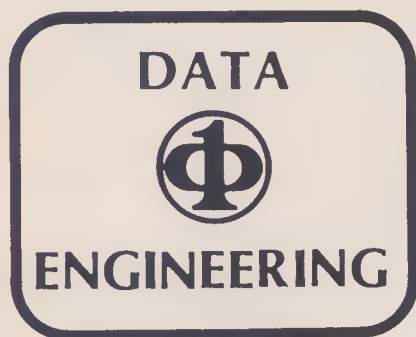


Joydeep Ghosh is currently a candidate for a PhD at the University of Southern California, where, since 1983, he has been the recipient of the highest University Predoctoral Merit Fellowship awarded by USC. He obtained a B.Tech degree in electrical engineering from the Indian Institute of Technology, Kanpur. His research interests are in the areas of computer architecture and parallel processing.



Raymond Chowkwanyun graduated with a BS in mathematics from the California Institute of Technology and an MBA in finance from the Wharton Graduate Division of the University of Pennsylvania. He recently completed an MS in computer engineering at the University of Southern California and is now working towards his PhD. His research interests are in Lisp- and Prolog-based AI machines.

The Third International Conference on Data Engineering



February 2-6, 1987
Pacifica Hotel
Los Angeles, California

ADVANCE PROGRAM

Sponsored by the Computer Society of the IEEE

Database Engineering is concerned with the role of data and knowledge about data in the design, development, management, and utilization of information aspects of databases, knowledge bases, and data management in general. The purpose of this conference is to continue to provide a forum for the sharing of experience, practice, and theory of automated data and knowledge management from an engineering point-of-view. The effectiveness and productivity of future information systems will depend critically on improvements in their design, organization, and management.

GENERAL SESSIONS: February 3-5, 1987

- Access Methods
- Distributed Operating Systems and Distributed Databases
- Database Design and Implementation
- Performance Evaluation
- Relationship Between Data Engineering and Software Engineering
- Architectural Support for Database Management
- Evaluating Recursive Queries
- File Structures
- Parallel Processing in Database Systems
- Object Based Systems
- Performance in Distributed Systems
- Statistical Databases
- Symbolic Processing
- Improving Concurrency in Distributed Systems
- Fault Tolerance and Correctness
- Knowledge Representations
- Resiliency in Distributed Systems
- Fault Tolerant Storage Systems
- Data Modeling
- Experiences in Data Engineering I
- Database Security Issues
- Historical Databases
- Experiences in Data Engineering II
- Engineering and Information Manufacturing Systems
- Extending the Relational Model
- CAD/CAM Systems
- Query Processing

PLENARY SESSIONS

The Adventures of a Real-World Database Machine Company, P. Neches

Uncertain Data Management, L. Zadeh

Personal Database for Personal Computers: Intelligent & Flexible Databases for Individuals, J. Kaplan

What Have We Learned?, B. Wah

TUTORIALS: February 2, 5 and 6, 1987

Monday

9:00 am-5:00 pm: Computers for Artificial Intelligence Processing, Benjamin Wah

9:00 am-5:00 pm: Real Data Engineering, Tom Gilb

9:00 am-12 noon: An Introduction To NIAM Modeling, Paul S. Thompson

9:00 am-12 noon: Reference Architecture for Distributed Database Management Systems, Saeed Rahimi

1:30 pm-4:30 pm: Distributed Database Management Principles, Saeed Rahimi

6:30 pm-9:30 pm: Automated Support Environments for Database Design, David Reiner

6:30 pm-9:30 pm: Heterogeneous Distributed Databases: Issues in Integration, Amit P. Sheth

Thursday

6:30 pm-9:30 pm: Introduction to Object Oriented Concepts: Smalltalk and Database, James Diederich and Jack Milton

Friday

9:00 am-12 noon: Logic Programming, Expert Systems and Databases, Steve Hardy

9:00 am-5:00 pm: Database Application Development Tools, S. Bing Yao

For a complete Advance Program, contact:

Data Engineering Conference
c/o Computer Society of the IEEE
1730 Massachusetts Ave., N.W.
Washington, DC 20036-1903
(202) 371-0101

The Third International Conference on Data Engineering

DATA ENGINEERING REGISTRATION FORM FEBRUARY 2-6, 1987

Complete and return this form with your check made payable to DATA ENGINEERING CONFERENCE to:

Data Engineering Conference
c/o Computer Society of the IEEE
1730 Massachusetts Ave., N.W.
Washington, DC 20036-1903

	Member	Non-Member	Student/Retired
Conference	\$190	\$240	\$35
Half Day Tutorial Fee	90	110	
Full Tutorial Fee	160	200	
		Late Registration After 1/9/87	
	Member	Non-Member	Student/Retired
Conference	\$230	\$290	\$35
Half Day Tutorial Fee	110	135	
Full Tutorial Fee	200	250	

Conference Amount: \$ _____
Half Day Tutorial Fee: _____ x \$ 90/\$110/\$135 = \$ _____
Full Tutorial Fee: _____ x \$160/\$200/\$250 = \$ _____
TOTAL ENCLOSED: \$ _____

PLEASE CHECK THE APPROPRIATE TUTORIALS YOU ARE ATTENDING:

- Tutorial #1 B. Wah ☐ (Full Day)
Tutorial #2 T. Gilb ☐ (Full Day)
Tutorial #3 P. Thompson ☐ (Half Day)
Tutorial #4 S. Rahimi ☐ (Half Day)
Tutorial #5 S. Rahimi ☐ (Half Day)
Tutorial #6 D. Reiner ☐ (Half Day)
Tutorial #7 A. Sheth ☐ (Half Day)
Tutorial #8 J. Diederich/J. Milton ☐ (Half Day)
Tutorial #9 S. Nardy ☐ (Half Day)
Tutorial #10 S. B. Yao ☐ (Full Day)

IEEE or IEEE Computer Society Membership No. _____

Name _____ Organization _____
Street Address _____
City/State/Zip _____ Phone _____
Credit Card Name _____ Credit Card No. _____ Expiration Date _____
Cardholder's Name _____ Signature _____

To guarantee your reservation, either a check or appropriate credit card information (Visa, MasterCard, Choice, or American Express), must be included with the conference registration form.

Notes: Conference registration fee includes admission to the technical sessions, one copy of the conference proceedings (except students), break refreshments, and reception. Refunds (less \$15 handling fee) will be made if request is received in writing by January 19, 1986. Tutorial Registration includes tutorial notes and lunch to those who attend a full day tutorial or a 9 am-12 noon and a 1:00-5:00 pm tutorial on the same day.

DATA ENGINEERING HOTEL RESERVATION FORM

Hotel Address: Pacifica Hotel and Conference Center
6161 Centinela Avenue
Culver City, CA 90230-6306
Tel.: (213) 649-1776
1-800-421-1448 or 1-800-262-1574
(California only)

The hotel is located 3 miles north of Los Angeles International Airport and provides 24-hour complimentary limousine service to and from Los Angeles International Airport. The bus runs every half-hour. Use the telephone in the baggage claim area to call the hotel for pickup.

Rates: Single: \$75.00 + 8% tax
Double: \$78.00 + 8% tax

Govt. Rates: Single: \$52.00 + 8% tax
Double: \$62.00 + 8% tax

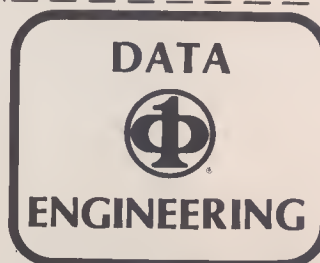
Credit Cards: All major credit cards accepted

To be able to confirm the reservation, we must receive the coupon by **January 19, 1987**. To guarantee reservations, we also must receive one night deposit, plus 8% occupancy tax.

Reservations made after January 19, 1987 are subject to space availability. Check-in time is 3:00 p.m. Check-out time is 12:00 noon.

Cancellations must be received by hotel at least 48 hours prior to date of arrival.

Name _____
Street _____
City/State/Zip _____ Phone _____
☐ Single ☐ Double
Arrival Date/Time _____ Departure Date _____
Credit Card Name _____
Credit Card Number _____ Exp. Date _____
Cardholder Name _____
Signature _____



**THE COMPUTER SOCIETY
OF THE IEEE**

**THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.**
IEEE

Software Development Support for AI Programs

C. V. Ramamoorthy, Shashi Shekhar, and Vijay Garg
University of California at Berkeley

In spite of the widespread commercial use of expert systems, vendors cannot guarantee the quality of the programs.

Artificial intelligence is a growing branch of computer science that studies ways of enabling computers to do tasks that seem to require human intelligence. These tasks include game playing, expert problem solving, natural language understanding, and theorem proving. Many existing AI programs perform such tasks with varying degrees of success. Whereas a program that can understand natural language is still a dream, many chess-playing programs can beat expert human players.

More successful AI programs include knowledge-based expert systems that are being applied to a wide spectrum of real-life problems from airline catering to oil exploration. These systems can acquire knowledge about a domain from a human expert and then use it to solve routine problems in that area. For example, expert systems can perform almost as well as human experts in the diagnosis of infectious diseases (MYCIN), finding the struc-

ture of chemical compounds (DENDRAL), and performing mathematical symbol manipulations (MACSYMA). These AI systems offer new capabilities to tackle several problems difficult to solve using the conventional algorithmic approach.

However, some difficulties exist in using AI techniques for solving real-world problems. The AI programming style differs markedly from that of conventional languages such as Fortran or Pascal. Because of this, we face a shortage of AI programmers and experts. The problem is aggravated by the fact that each area of AI has its own programming style, often unsuitable for other applications of AI. Also, the development of AI programs typically requires a software development environment that supports all phases of the software life-cycle: requirements specification, design, implementation, testing, and maintenance. Existing AI development environments support just the implemen-

Table 1. What makes AI programs different.

Feature	AI Programming	Conventional Programming
Processing type	Symbolic	Numeric
Technique	Heuristic search	Algorithmic
Definition of solution steps	Not explicit	Precise
Answers sought	Satisfactory	Optimal
Control/Data separation	Separate	Intermingled
Knowledge	Imprecise	Precise
Modification	Frequent	Rare

tation phase, which results in less reliable and hard-to-maintain programs.

In this article, we discuss software development environments for AI programs to deal with these difficulties. An ideal software development environment should provide a powerful language for the application, as well as appropriate methodology and tools for program development. This article introduces popular AI languages and describes how their features have been combined in expert system shells to provide a powerful language. We also present an overview of the evolution of tools for AI programming. The evolution is divided into five classes: discrete tools, the toolbox, life-cycle support, knowledge-based tools, and the intelligent life-cycle support environment.

Present commercial AI program development support environments fall into the toolbox class, and they will have knowledge-based tools and life-cycle support for AI programs in the near future.

Inside an AI program

AI programs differ from conventional algorithmic programs. The latter often have a fixed sequence of steps defined precisely by the programmer, whereas AI programs, like human beings, often take a trial-and-error approach to finding a satisfactory solution. Table I lists some of the differences between conventional programs and AI programs.

The essence of AI lies in encoding knowledge about a domain and then using it to solve problems in that domain. This task is usually broken into three parts: *knowledge representation* for encoding information about the domain, *pattern matching* for retrieving knowledge relevant to the problem, and *searching* for a satisfactory solution among a large number of possibilities.

We now discuss these basic components of AI programs.

Knowledge representation. The amount of knowledge needed for any interesting AI application tends to be very large, and therefore to access, update, and maintain such large amounts of knowledge, AI programs need good knowledge representation schemes. AI researchers have used Prolog clauses, semantic networks, frames (see Figure 1) and many other schemes for representing knowledge, each having its advantages and disadvantages. For example, a semantic network keeps the related information together, but has difficulty in-

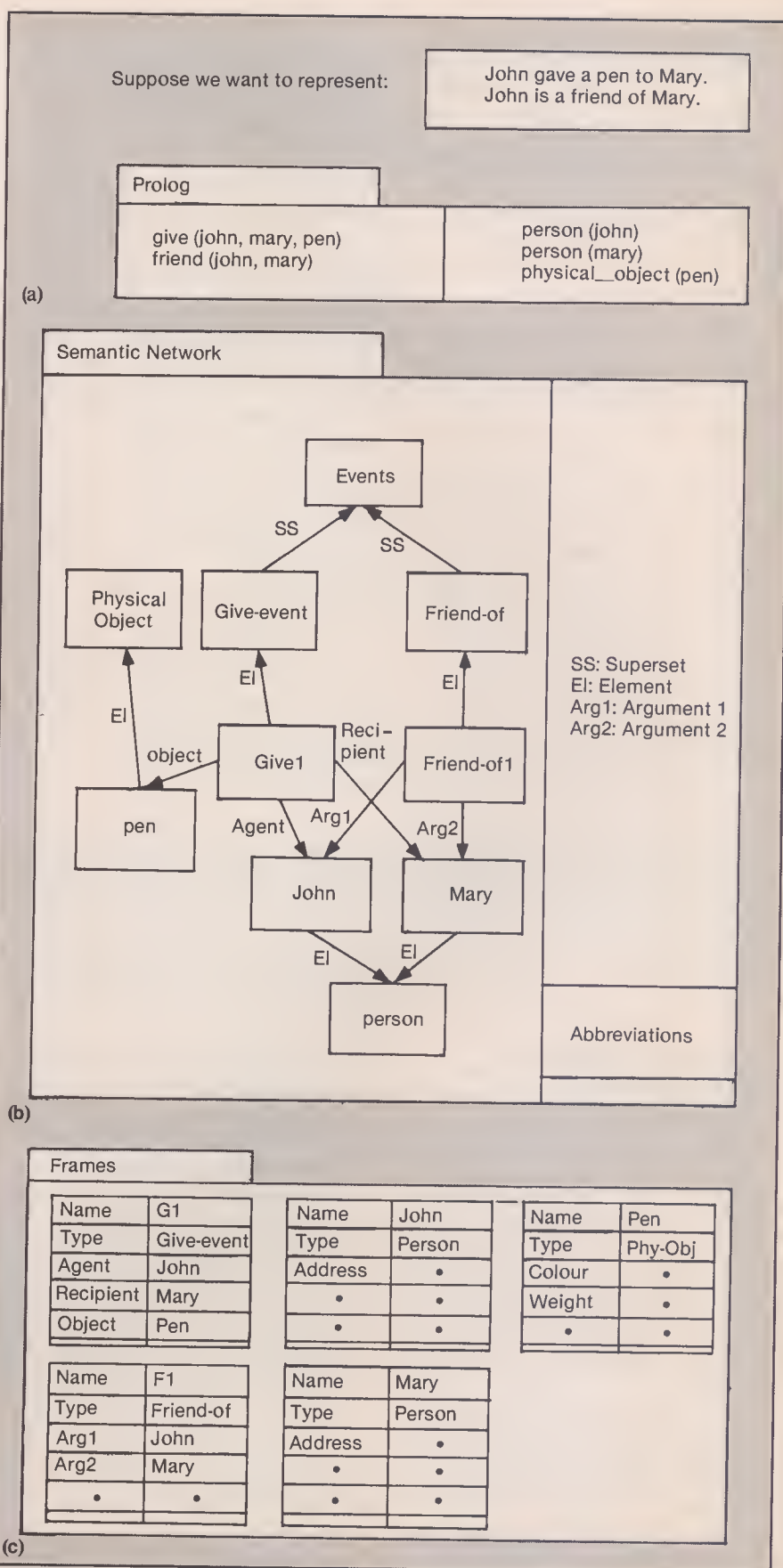


Figure 1. Knowledge representation schemes: (a) Prolog, (b) semantic network, (c) frames.

ferring from it. Prolog, on the other hand, has mechanisms for logical deduction, but does not keep related information together.

Most traditional AI schemes, however, are suitable only when all the knowledge is in main memory. If the knowledge base grows too large to fit in the main memory, then disk-based databases have to be used. The design of efficient databases to store rules, objects, frames, etc. is still in the research stage.

Pattern matching. AI language interpreters use sophisticated pattern matching for selecting relevant rules at any stage of problem solving. Prolog systems match (unify) the current goal against various facts and rules in the database for invoking pertinent rules. The unification algorithm checks for equality of atoms and bound variables and tries to bind free variables to the value being matched against. More sophisticated matching, as in OPS5, allows condition evaluation to check if the current state of global memory satisfies the specified condition. The conditional expressions can include relational operators such as equality and less-than, as well as connectives such as AND, NOT, and OR.

Searching. AI programs choose from the set of all possibilities (search space) the most promising one to pursue for the solution. To reduce the amount of search, AI programs use heuristics. Like rules of thumb, they often succeed but may fail occasionally.

AI programming paradigms and languages

A programming paradigm is a set of related concepts useful in conceptualizing, analyzing, and solving a problem. A programming language is a medium in which one or more paradigms can be expressed. For example, object-oriented programming is a paradigm, whereas Smalltalk is a programming language based on that paradigm.

In this section, we summarize some popular AI programming paradigms by presenting for each a programming language based on that paradigm. We also discuss efforts to provide a homogeneous combination of multiple paradigms in a single programming language. Finally, we discuss some new developments that will affect the design of future AI languages.

Lisp: Functional paradigm. A Lisp program consists of independent functions, each a mapping rule that translates its arguments to a result. These functions are created by composing system functions or user-defined functions using control structures such as *if* and *recursion*. System functions include list processing primitives such as *car*, *cdr*, *member*, and *append*; mathematical primitives such as *add*, *times*, and *division*; and Boolean predicates such as *and*, *negation*, and *equal*.

Another underlying structure used in Lisp is *list*. Lisp provides functions to concatenate, build, extend, and dissect lists. Some examples of lists are (ON TABLE BOOK) and (ADD 3 (ADD 4, 8)).

Another example of a Lisp function is given in Figure 2a. Because both data and functions are represented as lists, one can write Lisp functions to manipulate other functions.

There are a number of versions of Lisp in use today. MacLisp, developed at MIT, is noted for efficiency, while Interlisp of Xerox Corporation has the most sophisticated program development environment. Common Lisp and Franz Lisp are other popular versions. We will take a look at the Interlisp environment later.

Prolog: Logic programming. Logic programming is usually based on a theorem prover, which can tell us if a given set of logical formulas has any contradiction or not. A Prolog program may be viewed as a collection of formulas in propositional logic with a theorem (query) to be proved. Formulas are restricted to implications, where each side of an implication is a restricted logical expression. They are used to represent various facts (data) and rules of implication. An example of facts and rules is given in Figure 2a.

Prolog provides a database facility to store facts and rules for efficient manipulation. A Prolog interpreter uses pattern matching, called unification, to select relevant rules from the database to answer a particular query. The user does not have to specify the exact sequence of steps to carry out a computation. This mechanism helps in programming natural language interfaces because rules of language grammar fit easily into Prolog syntax. Because Prolog is younger than Lisp, none of its program development environments are as good as that of Interlisp.

OPS5: Production systems. An OPS5 program consists of two parts: a global memory and a set of rules. The global

memory can store data structures, which are referenced and manipulated by rules. Each rule has a left-hand side composed of a set of condition elements and a right-hand side that specifies actions (see Figure 2b). If all the condition elements of a rule match the contents of global memory, then the rule is applicable. The OPS5 interpreter in a recognition-action loop selects all applicable rules and uses a criterion known as the *conflict resolution strategy* to choose one rule from among them. This strategy often depends on execution time parameters, thus making it difficult to predict the behavior of the program statically, but it has the advantage of making unplanned responses possible.

Production system languages such as OPS5 are generally unsuitable for coding algorithms, since it is difficult to do iteration and recursion in them.¹ Also, OPS5 rules are independent of each other and cannot be grouped together as a set of related rules in a module, making the development of large programs difficult.

Smalltalk: Object-oriented style. In this paradigm, a program consists of objects and messages. This way of programming brings modularity in the program by data abstraction and is especially suited to those situations with a clear hierarchical classification of objects.

Smalltalk introduces two new concepts in programming: data abstraction and inheritance. Data abstraction refers to the encapsulation of the data-structure definitions with procedures that can directly manipulate the representation of the data. All other procedures have to send a message to this object to access or update the data stored in it.

The concept of inheritance is useful in representing knowledge about a hierarchical classification. For example, consider the taxonomy used by biologists for living things. The class of humans is a subclass of mammals, hence it suffices to store the properties related to mammals in one place, with the understanding that all instances of humans will inherit the properties of the superclass mammals. This representation also avoids inconsistencies arising from duplication of the same information in several places.

Others. Several other programming models are used by researchers in AI. The augmented transition network parser used in natural language research implements grammar rules for the English language. The ATN parsers are top-down parsers

with the facilities of special variables, conditions, and actions to aid in programming.

A program for this system consists of a set of grammar rules and associated statements to be carried out when a grammar rule is applied. *Frame-based systems* have some features of the object-oriented style as well as those of production systems. Each frame represents knowledge about a stereotypical situation. During execution, the current state can be matched against one part of a frame to check if that frame is applicable. The frame consists of an action to be taken and other conditional checks to see that the action has been performed correctly. The paradigm has been used successfully in understanding stories about certain stereotypical situations.

Towards an amalgam

Each of the several programming languages in AI was designed with a set of specific problems in mind. Each features some interesting and useful constructs for helping to solve those problems. For example, Prolog and ATN are more convenient for developing parsers for a subset of English than, say, Lisp.

This multiplicity of languages, however, poses several problems. It makes it difficult for people to share their work and development tools. They also have to learn a new language whenever they try to explore a new problem domain. In view of this, recently three commercial systems, namely KEE, ART, and LOOPS, have tried to combine some features from objects, frames, rules, etc. They provide an integrated set of tools to allow multiple paradigms in the same program. These systems, called *expert system shells*, concentrate on the programming needs for building knowledge-based expert systems.

KEE, LOOPS, and ART. The Knowledge Engineering Environment² from Intellicorp is a frame-based system built on top of Lisp. Frames represent both objects and rules and can be linked together to group them in a hierarchical manner. Frames consist of slots describing attributes of objects, constraints on their values, source of inherited values, procedure to be invoked when these attributes are accessed or updated, and rules affected by those objects. Lisp functions represent procedures, and predicate calculus represents production rules and relations between various objects. Figure 3 shows the

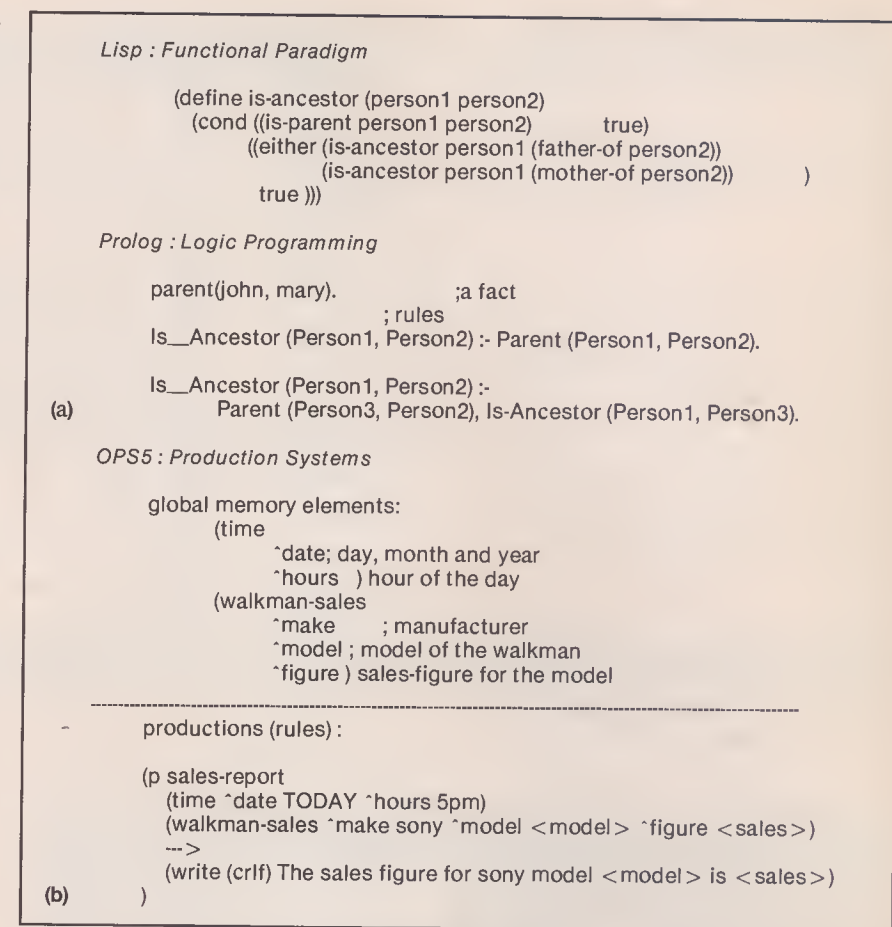


Figure 2. Glimpses of various programming languages. (a) The is-ancestor function takes two arguments and returns true if Person1, the first argument, is an ancestor of Person2, the second argument. The Prolog program for the same purpose would do something extra. It can also find-out all ancestors of Person2, or all descendants of Person1. (b) This sales-report rule will be activated at 5pm every day. It will print the sales figure for all models of Sony walkman. The names surrounded by angle brackets (< and >) are variables and the names prefixed by ^ are slot-names.

knowledge base for a particular process control application. This hierarchically-organized knowledge base has a mix of rules, frames, etc.

A deductive retrieval facility named TellAndAsk³ provides a Prolog-like interface for accessing the knowledge base. Prolog-like logic programming is also used for rules interpretation. The rule system provides some tools for debugging and explanation. A *rule class* graph shows conflicting rules that can be invoked for the same goal. A *How* graph shows the derivation tree for any conclusion.

LOOPS (from Xerox PARC),^{4,5} a system built on top of Interlisp-D, combines the functional paradigm of Lisp, the object-oriented style, and rules. LOOPS represents data structures and rules as objects. It introduces the concept of composition of rules by allowing a rule to call other rules directly or by sending mes-

sages. Similarly, objects can be combined or grouped together to form higher-level objects. Procedures defined inside an object can be rules or Lisp functions. Interlisp tools have been upgraded to provide a uniform set of tools for the mixed paradigm.

The ART system⁶ from Inference Systems combines frames, rules, and processes. It represents facts by logical assertions or frames. For activities involving inferences and consequences, it uses a rule-based paradigm.

ART also introduces the concept of a process that can incorporate rules and procedures with internal control information for independent computations.

The future. Present efforts toward integrating useful concepts of all paradigms can be useful only as short-term measures. Various important developments in AI

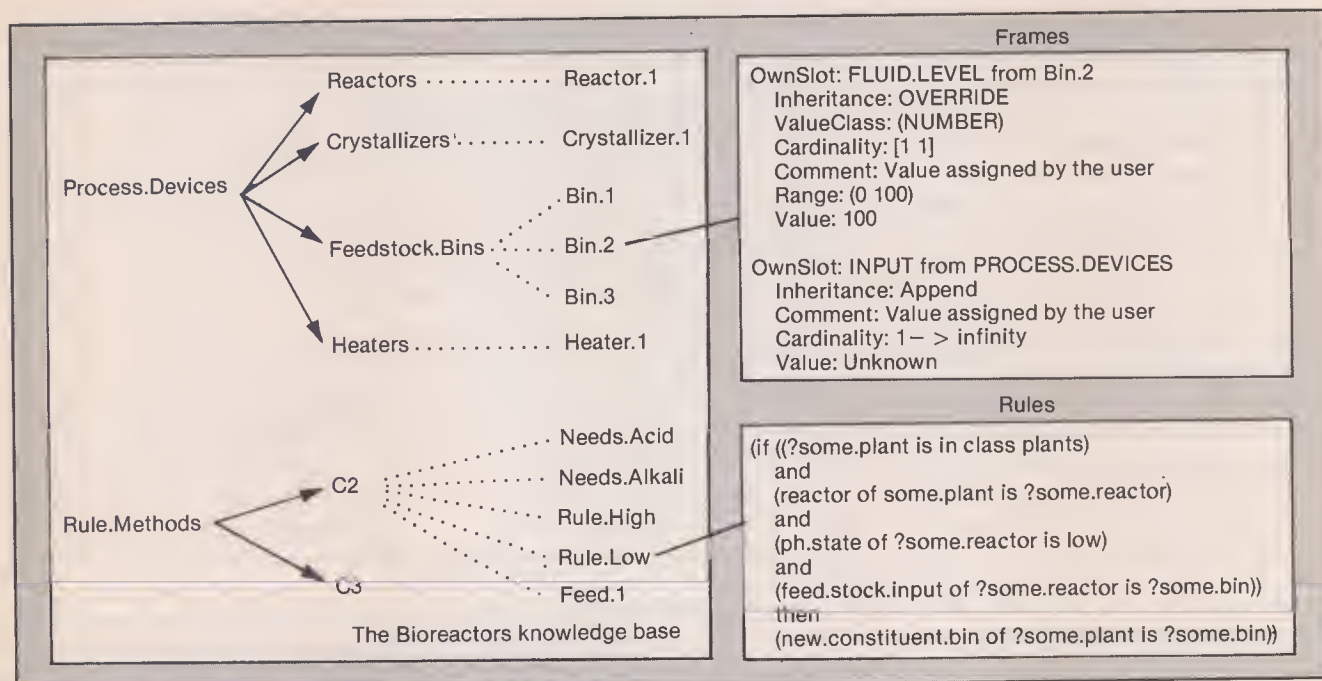


Figure 3. A typical knowledge base in KEE. (Source: "The Knowledge Engineering Environment." Used by the courtesy of IntelliCorp, Knowledge Systems Division.)

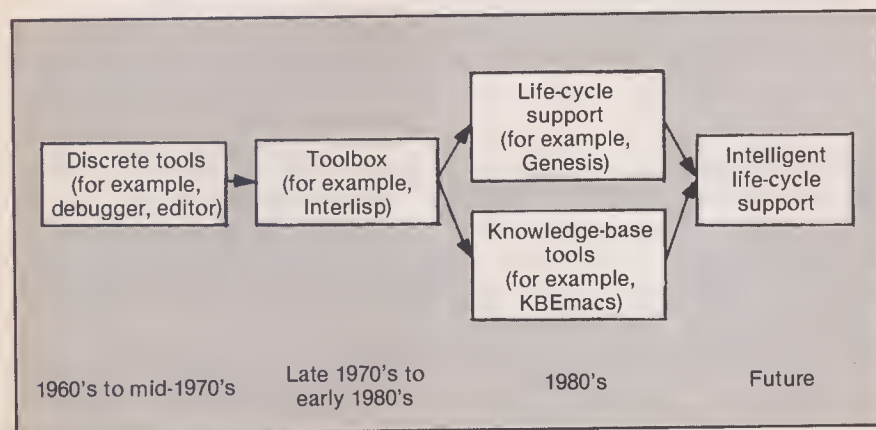


Figure 4. Classification of development environments.

need to be incorporated in these amalgams. Some of these developments include representation of belief, cooperating distributed AI systems, and disk-based AI systems.⁷ The representation of belief helps in decision-making situations, where some of the evidence is in favor of a hypothesis and some against. The uncertainty of this situation is quite common in real world problems such as medical diagnosis (MYCIN) and oil exploration (Prospector).

It is difficult for a tightly integrated system to incorporate all the new features. We would rather see an open architecture, which can accommodate newer paradigms, as the choice of the future.

Development support for AI programs

The purpose of development support is to facilitate the creative and intellectual step from the conception of an idea to its description in a computer-understandable form. Peculiarities of tools, programming languages, and user interfaces often present a formidable problem to the user: The user has to spend more time translating an idea into a computer language than polishing the idea to improve it. This situation is unfortunate, and development support should try to relieve the user from as many

details as possible and permit concentration on the idea.

The evolution of development support technology can be classified into five distinct parts as shown in Figure 4.

Tools. Development tools have been around for a long time. Translators of programming languages and debuggers to monitor the execution-time state of programs were some of the first tools. Debuggers, along with screen editors, remain the most useful tools even now.⁸ Other popular tools include pretty printer, configuration control support, program cross referencer, and execution profiler.

These tools were discrete and independent. The programmer could access only one tool at a time. For example, during debugging you had to use the debugger to identify a bug and then terminate debugging to invoke the editor for fixing the bug. After this, you would have to compile your program and call the debugger for further debugging (the edit-compile-run cycle). It forced the programmer to type in extra commands and wasted time in starting and terminating system tool execution. Interrupting the debugging process to edit, recompile, and rerun the program severely hampered the thinking process of the programmer.

Toolbox. The next development in support environments was an integrated set of

tools called the *toolbox*. Each tool was designed with others in mind, therefore the system provided features to access other tools from inside the executing tool.

For example, Interlisp allowed a programmer to run a function (program) right after typing it into the system. In response to an error, the system invoked the debugger to allow the programmer to investigate the cause of trouble. The programmer could then edit the function and continue execution. This approach reduced the time needed for fixing minor errors in the program for the exploratory programming (trial and error) commonly used by the AI community.

The Unix system provided a relatively loosely-integrated set of tools for a wider range of functionality. The editor could be called from inside the debugger, but the programmer still had to recompile and rerun the program. Unix also presented a wide array of tools for data manipulation (grep, wc, head, more, awk, join etc.), rapid prototyping (csh, awk, pipe, I/O redirection), configuration management (make, ar, awk) and many other such routine operations.

Life-cycle support in software engineering. The design of a large software system is a complex and difficult problem. The division of its life-cycle into several phases (requirement analysis, specifications, design, implementation, testing and debugging, operation and maintenance⁹⁻¹¹) is aimed at reducing the complexity of design by isolating and ordering the important tasks in the development process.

We discuss each phase of the life-cycle later. Here we simply want to note that Unix and Interlisp support only the implementation and debugging phases. Studies show that the largest cost of the life cycle comes from the maintenance phase. Similarly, the requirement and specification phase tools are important, since any errors in specifications may be very difficult and expensive to correct in later stages of the life-cycle. Thus, life-cycle considerations are important for any piece of software that is going to be used for an appreciable length of time by end users. Although the life-cycle model for most experimental AI programs differ, the issues remain somewhat similar.

Few programming environments provide support for the entire life-cycle. Gandalf,¹² Genesis,¹³ and Saga¹⁴ represent efforts in this direction.

Intelligent tools. The recent success of AI techniques in several areas motivated the use of AI techniques in the development of software. The Programmer's Assistant project at MIT and the Psi project at Stanford University are two representative systems. These projects try to model the knowledge a programmer uses in understanding, designing, implementing, and maintaining a program. This knowledge can be used by expert systems to partially automate the program development process.

AI techniques are also being used to automate certain optimization decisions, like data-structure selection. We will describe this approach in more detail later.

Future intelligent life-cycle support. Program development environments in future will provide knowledge-based tools for life-cycle support. AI techniques will support different phases of program development. The Rome Air Development Center Report¹⁵ described knowledge-based tools for developing and validating requirements for any user program. The fifth generation project¹⁶ includes plans to provide intelligent life-cycle support.

We now follow our abstract view of support systems with a look at reality.

Tools from Prolog environments

This section describes some of the development tools available for Prolog programming. The tools have been taken from various versions of Prolog, including the Prolog-10 family,¹⁷ Poplog,¹⁸ Micro-Prolog,¹⁹ and APES.²⁰

Explanation. The explanation of any conclusion (result) derived by the system is the sequence of rules and facts used to reach the conclusion. The explanation is like an argument in favor of the conclusion, and of course the user may or may not agree with the argument.

The argument can be used effectively for debugging the knowledge base of rules and facts. For example, the explanation facility presents a better user interface for a doctor using an expert system for the diagnosis of a disease. Because the expert system explains its conclusions, the doctor remains in control and can accept or overrule the machine. The explanation of APES and *why* of Micro-Prolog are examples of such facilities.

In some sense, explanation is like the execution trace of the program computing a result, but it is quite useful if the number of rules and facts applied is relatively small. APES displays an edited form of execution trace showing how rules and facts were used.

Tracing and debugging. The Prolog-10 family provides a trace package based on the *Box model*, which views the execution of a Prolog program as a sequence of call, exit, fail, and redo events. Prolog program execution starts when the user types in a query. The Prolog interpreter selects a rule, which can answer the query or break the query into simpler queries.

The selection and invocation of a rule is called a *call*. If the query is answered, the invoked rule exits successfully. Otherwise, the invoked rule fails to satisfy the query and other applicable rules are tried (redo the query). The user can place watchdog points at the four events (call, exit, fail, and redo) to trace the execution.

Editor and filing system etc. Poplog provides a screen-oriented editor with limited windowing facilities. The editor is integrated with the debugger. Either can be accessed from inside the other.

The editor also has a documentation facility for user programs. The documentation is stored at three levels: Help, Teach, and Reference. An associated program library can be automatically linked to the user programs. It can also be accessed and modified by the user.

Micro-Prolog provides an English-like syntax for entering Prolog facts and rules into the system. For example, *mother (mary, sue)* can be paraphrased as *Mary is the mother of Sue*.

Many versions of Prolog provide metavariables. A metalevel Prolog program is capable of processing other Prolog programs. This facility makes it easier for the user to write his or her own development tools in Prolog itself. Small tools such as pretty printers and experimental Prolog interpreters with various chaining/evaluation strategies can be written in Prolog itself.

The Interlisp environment

Interlisp, or Interactive Lisp, is an integrated set of tools, or toolbox, for developing programs in Lisp. Developed about ten years ago, it provided one of the most

sophisticated forms of development support at that time. It included uniform error handling, automatic error correction, a debugger, a filing system, and a screen-oriented editor.^{21,22} Lisp was enriched with new constructs to access these tools through function calls.

The tools are integrated in two ways:

(1) All the tools are at the programmer's disposal in almost every state of computation. For example, the programmer can access the editor from inside the debugger and then resume debugging.

(2) A single tool is used for one purpose in all states. For example, the automatic error-correction module is used in all other tools for error correction.

Now we will describe the individual tools.

Do what I mean (DWIM). The command interpreter of most tools is intolerant of trivial errors by its user. Often a small spelling mistake requires the user to type in the entire command again. DWIM is the automatic error-correction facility invoked by the system whenever the interpreter detects an error. The DWIM tries to guess what the user might have intended and fixes the mistake accordingly. It seeks the closest match to the misspelled word in the collection of words recently used by the user. It can take care of spelling mistakes in the command languages of the editor, the debugger, and the file system.

DWIM also takes care of many other syntactic errors of this kind. Unfortunately, the corrections are not always right, which can cause execution of undesired commands.

Masterscope. During the debugging process, the programmer often wants to look at definitions of particular functions and variables in a program. If the program is small, a hardcopy printout or the editor window may be adequate, but if it is large (say, a 10,000-line program scattered over several files and directories, with scores of functions and variables), then retrieval of particular definitions requires a substantial effort.

To make this task of retrieval easier, Interlisp has a tool called Masterscope, consisting of a database of program-related information, a program syntax analyzer (parser) to build the database, and a query language for the database. Masterscope lets the user retrieve the declaration of any program object, such as a function, a variable, or a type, with little effort.

Programmer's Assistant. The interactive environment offers new ways to provide a more friendly and helpful design environment to users. A user working on a new program often tries an exploratory approach to design or debug a module. He or she might modify a file and later want to revoke the decision if the modifications do not yield desirable results. The Programmer's Assistant provides a facility for this kind of exploratory programming. It maintains a *history list* of all the commands issued by the programmer with the side-effects (changes in functions, directory, etc.) they caused. The list is used to revoke (undo) a previous command, if the user so requests.

Similarly, the Programmer's Assistant lets the user repeat (redo) a sequence of past commands. Old commands can be composed together and then modified to form a new command that saves typing effort and time.

File system, editor, and debugger. On any modification, the file system loads, recompiles, or creates a backup of relevant files. It gives the view that the entire program resides in a database and that functions, variables, etc. can be accessed conveniently through a powerful query language.

The editor can be called by a user command or from inside an executing function. Users can add new commands by adding their own functions to the editor module. The debugger is invoked automatically when the executing user program has a major error. The debugger enters a read-eval-print loop and lets the user execute an arbitrary function as a debugging command. The functions can look at the back-trace or at the present state of variables, change the present state, invoke the editor to change a function, and resume execution.

Windows and graphics. The Xerox 1108 Interlisp-D System uses a large format display and a mouse pointing device to allow better computer-user interaction. A window system lets the user divide the workstation screen into several logical screens (rectangular areas called windows). Each window is like an independent screen to view a file or to execute a command, and it can display text, graphic images, line drawings, or a combination of all these. A mouse is used to select one window out of the existing windows to work in.

Life-cycle support for AI programming

Because of the increasing use of expert systems in commercial applications, they should be reliable and robust. Software engineering has provided concepts for developing reliable programs in conventional applications. It divides the life of a program into phases: requirement analysis, specification, design and implementation, usage and maintenance. The division of the software life-cycle into various phases reduces the complexity of design by grouping and ordering the main tasks of program development.

Requirement analysis is aimed at getting a complete, consistent, and unambiguous idea of the needs of the user, which can also serve as a common ground for agreement between the customer and the developer. The results help chalk out the specification of the software system, against which the software product can be developed and tested.

The implementation, testing, and debugging phases need no elaboration.

The maintenance phase consists of software updates as well as repair of newly discovered bugs.

Requirements engineering. AI program development should begin with writing complete, consistent, and unambiguous requirements of the system. The main advantages of writing requirements follow:

(1) It helps in early detection of errors. Studies²³ have shown that 30 percent of errors are due to faulty specification of requirements and are 10 to 100 times more expensive to fix at the implementation stage than at the requirements stage.

(2) It helps in the detection of misunderstanding between the customer, the developer, and the user.

(3) If requirements are written formally, then expert system programmers can check that all the requirements of the system are met. This would be difficult to do otherwise.

(4) It helps in generating good test cases and judging the quality of the test cases actually used.

Design concepts. Design of the system should use software engineering principles such as information hiding, separation of concerns, layering, and modularity. The principle of *information hiding* can guide us in designing rules. It suggests that rules and their group should be designed such

that they hide some internal details about the system. For example, if a queue is used in the system, then its priority mechanism should be hidden from rest of the system.

While designing rules, care must be taken that different functionalities are separated and different rules implement these functionalities. For example, rules governing pressure and temperature in a control system should differ from rules governing security of the system. This is the principle of *separation of concerns*.

The principle of *layering* suggests that the system should be designed as a composition of layers. Any layer is aware only of the layers underneath it. Thus, activation of rules at a higher level due to activation of rules at a lower level should be minimal. Such a design simplifies not only implementation but also testing.

The principle of *modularity* suggests that rules that access common information and implement related functionality should be clustered into one module for easy implementation and maintenance.

Testing by prototyping. Testing of AI systems is different from that of conventional systems. Firstly, AI systems often show nondeterministic behavior because the *conflict resolution strategy* can depend on execution-time parameters. This makes the behavior unrepeatable and therefore more difficult to debug.

Secondly, there is no precise input-output relationship for rules as there are for procedures in conventional software. This makes it difficult to use input-output analysis for testing.

Thirdly, the number of ways rules can be activated is too large to use branch- and path-coverage tools.

Prototyping is the only effective way to test an AI program. User experience with the prototype can reveal vital problems in the design. However, prototyping of a large system can take substantial effort and time.

Maintenance. Maintenance and modification is an important part of expert systems development. Rules evolve with the experience of their use and are therefore modified more often than algorithms. In the case of inexact reasoning, the belief support of rules varies with the feedback from previous support numbers. Rules can also be time-dependent, hence their validity can change over time.

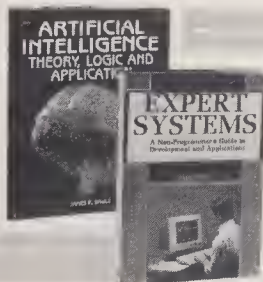
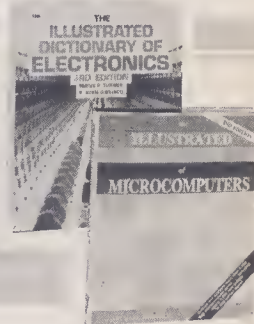
Unfortunately, not much attention has been paid to maintenance in expert system programming, although many useful tools

An Absolutely No-Risk Guarantee

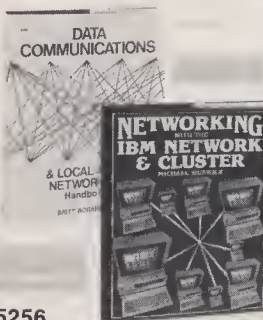
Choose one of these five valuable selections for only \$2.95 . . . as your introduction to The Computer Book Club!

Here's your opportunity to save on outstanding books covering the most recent developments in computer science, data processing, artificial intelligence, and robotics! We can help you keep in step with the most up-to-date technologies with the latest professional-level books and offer you an exceptional selection of proven bestsellers that have become classics in the field—all available in original publisher's editions at discounts of up to 50% off publishers' prices!

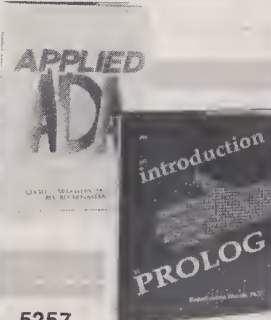
5259
The Illustrated Dictionary of Microcomputers—2nd Edition by Michael Hordeski and The Illustrated Dictionary of Electronics—3rd Edition by Rufus Turner and Stan Gibilisco. These two outstanding reference works have become the standards in their fields. Each provides instant access to thousands of essential terms and techniques reflecting the most current usage and terminology.
(2 vols.) \$59.90



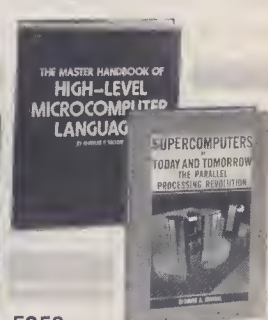
5255
Artificial Intelligence: Theory, Logic and Application by James F. Brule provides an in-depth overview of artificial intelligence from the history of its development to the latest state-of-the-art applications. Includes AI applications programs in BASIC and Expert Systems: A Non-Programmer's Guide to Development and Applications by Paul Siegel guides you through the four basic phases of expert system development and shows how to build an expert system for your own application needs. (2 vols.) \$48.90



5256
Data Communications and Local Area Networking Handbook by Britt Rorabaugh and Networking with the IBM® Network™ and Cluster™ by Michael Hurwicz.
(2 vols.) \$54.95



5257
Applied Ada® by Gerald L. Mohnkern, Ph.D. and Beverly Mohnkern and An Introduction to Prolog by Ramachandran Bharath.
(2 vols.) \$49.95



5258
Master Handbook of High Level Microcomputer Languages by Charles F. Taylor and Supercomputers of Today and Tomorrow by Richard A. Jenkins. (2 vols.) \$43.90



The Computer Book Club®

P.O. Box 80, Blue Ridge Summit, PA 17214
Or call Toll-Free 800-233-1128

Indicate by number the set you want.

Please accept my membership in The Computer Book Club® and send me the selection indicated, billing me \$2.95 (plus shipping and handling). If not satisfied, I may return the book(s) within 10 days without obligation and have my membership canceled. I agree to purchase 4 or more books at regular Club Prices (plus shipping/handling) during the next 12 months and may resign any time thereafter. I will receive the Club News 13 times per year listing hundreds of selections—mains, alternates, extras,—at savings of up to 50% off publishers' prices. If I want the main selection, it will be shipped automatically. If I want an alternate selection, or no book at all, I will notify the Club by returning the enclosed card. Starting immediately I will be eligible for the Club's Bonus Book plan offering savings of up to 80%. If not satisfied with any book, I may return it within 10 days without obligation.

Name

Address

City

State/Zip Phone

Valid for new members only. Foreign applicants will receive ordering instructions. Canada must remit in U.S. currency. This order subject to acceptance by The Computer Book Club®. IECC-187

for maintenance of conventional programs can be adapted for expert system programming. These tools include version management systems, configuration management systems, and modification request systems. A *version management* system is responsible for storage and easy retrieval of various versions of a program. A *configuration management* system is responsible for keeping the information about various modules of a program and generation of the program from these modules. A *modification request* system helps in estimating the impact of a change in the program.

Besides the use of these tools, it is also necessary that the program be designed keeping maintenance in mind. This involves proper documentation of all design decisions and constraints that affected these decisions.

Traceability. Traceability refers to the ability to locate in a database (of program-related information) objects of current interest. For example, if the user changes a particular requirement, traceability enables him or her to obtain all related specification and code modules that need modification.

Traceability can be as useful for large expert systems as it is for conventional software. For example, if the user changes a particular rule, traceability lets him or her retrieve all related rules that may conflict with the new rule.

Traceability can be provided using database technology: useful relations can be defined to link rules with each other, useful attributes can be defined to classify rules, and then rules can be retrieved using attribute or relation qualification. Examples of relations are *can_activate*, *semantically_related*, and *mutually_exclusive*. Examples of attributes are *author*, *date*, *relevant_requirement_section*, and *version*.

Knowledge-based tools

AI techniques deal with inherently difficult problems that cannot be solved without human assistance. Programming has been one of the tasks requiring a high degree of human intellectual involvement. Naturally, there has been a considerable effort to apply AI techniques to the programming process. People have worked on three different approaches to this problem: the transformational approach to program generation, modeling of the

programming process, and using AI techniques to improve other tools.

The transformational approach aims at generating software directly from its specification. This method can be viewed as an extension of the work in providing high-level language as a substitute for assembly-level programming. High-level languages provide relatively concise and more understandable notation for programs. They also delegate many lower-level decisions (such as register allocation and type checking) and implementation details (such as storage allocation and recursion) to a compiler and allow programmers to concentrate on the design process. The transformational approach can create a high-level specification language to relieve programmers of intermediate level decisions (such as data-structure selection and coding well-known algorithms) for various optimizations.²⁴

KBEmacs, a knowledge-based editor, illustrates some of the concepts described above.²⁴ KBEmacs provides *cliches* and *plans* for creating Lisp or Ada programs with few commands. A cliché is a generic parameterized procedure for performing a certain task (such as searching). The user can invoke a cliché and fill in parameters to use it as a part of the program. A good set of clichés can provide a vocabulary of relevant intermediate and high-level concepts for effective reasoning and communication. A plan provides convenient structures to represent the entire program for various manipulations by other tools.

We can separate the knowledge about a program into two parts: software engineering knowledge and application-specific domain knowledge. The software engineering knowledge is related to experience in software development (such as life-cycle models).

There are several efforts to develop a theory of software engineering knowledge—how expert programmers understand, design, implement, test, verify, modify, and document programs.^{24,25} This research can be used to automate some aspects of the program development process. The application-specific domain knowledge is particular to the problem. For example, the laws of mechanics and gravitation will be a part of the domain knowledge of a program guiding the space shuttle. This domain knowledge can be used for automatic checking of various programming decisions for conformance to the physical laws of the application domain. We can also use this knowledge to check the feasibility of requirements, and

for validation and testing of the design for correctness.

Various AI techniques can add new power to existing development tools. The RADC report described applications of AI techniques to acquisition, verification, and validation of requirements.²⁶ All too often, the user does not know precisely the features and capabilities he or she wants from the software. When that happens, the development of user requirements for a future software program resembles the knowledge acquisition process. Thus, knowledge acquisition tools can be used to obtain the requirements from the user.

Applications of AI techniques to software validation include theorem provers that can add the power of deduction to the database storing the program. Use of rule-based paradigms can make the prototyping of products quick and easy.

The Japanese Fifth Generation Project refers to knowledge-based development support as *systematization technology*.¹⁶ This project tries to use all three approaches to using AI for development support. The project aims to make it possible to synthesize programs from user requirements with automatic algorithm selection from an algorithm bank. It plans to achieve this by developing a theory for specification and verification of programs for automatic synthesis. Other goals of the project include the development of a consultant system for program design, and many other tools to maintain, improve, and manage programs. It is the most ambitious project in this direction to date.

AI programs are no longer mere experimental objects for a small group of researchers in universities: They are now being used for commercial purposes. Hence, development support tools are more necessary than ever to produce reliable programs in a cost-effective manner.

The sheer number of programming paradigms presents a difficulty in developing a standard set of development tools. They also challenge the creation of a single environment with powerful features of all the paradigms. Developments like KEE, ART, and LOOPS are a step in this direction. But there are still several issues unresolved in designing an open extendible architecture that can incorporate newer paradigms and features when needed. Presenting a simple and uniform semantic view of such a system to all programmers is a difficult design problem.



acm



CPS



CMCS

PEOPLE FIRST

CHI + GI 1987
SPECIAL COMBINED CONFERENCE OF
HUMAN FACTORS IN
COMPUTING SYSTEMS AND
GRAPHICS INTERFACE
TORONTO, CANADA, APRIL 5-9, 1987

HUMAN COMPUTER INTERACTION
Interface design and evaluation,
user interface management systems,
models and metaphors, training and
learning, systems in organizations,
input and output technologies...

INTERACTIVE COMPUTER GRAPHICS
Modelling, rendering, 3D interac-
tion, applications, and more...

PAPERS AND PANELS
The latest research results, recent
innovations and new directions, dis-
cussions and debates on the hottest
issues.

TUTORIALS
Learn from the leaders in a once-a-
year feast of leading-edge insights

DEMONSTRATIONS
Electronic theatre, videotapes, spe-
cial interest groups and informal
"poster" sessions

PEOPLE

Best of all, a world gathering of peo-
ple you ought to know in your field.
Join us!

Please send registration details for
CHI + GI 1987:

Name: _____

Organization: _____

Address: _____

Telephone _____

Return to: CHI + GI '87
2 Bloor St. W., Suite 100
Toronto, Ontario, Canada
M4W 3E2

IEEE

Sponsored by ACM Special Interest Group on Computers and Human
Interaction (ACM/SIGCHI) and The Canadian Information Processing
Society's Canadian Man-Computer Communications Society (CIPS/MCCS)
in cooperation with ACM/SIGGRAPH and The Human Factors Society.

AI programming has seldom concerned itself with the reliability and maintainability of the software, even though they are the most important considerations for software to be used for a long time. To make the products more reliable and maintainable, software engineering techniques must be used during development. These software engineering techniques include formal requirements specification, support for maintenance, and enforcement of a software life-cycle development methodology. The use of these techniques in the context of AI will become increasingly important in the future.

Using AI techniques can further increase the power of support tools. There has been some work in modeling the knowledge a programmer uses in understanding, designing, implementing, verifying, and maintaining programs. Based on this knowledge, an expert system can assist the programmer in performing many routine programming activities. It can also use the domain knowledge in checking the feasibility of requirements, validation of requirements, and design decisions, all of which can greatly reduce the need for inspection by human experts.

Besides the ways discussed here, advances in human factors, database design, networks, knowledge acquisition support, and evidential reasoning support will also lead to better software development environments for AI programs. □

Acknowledgments

We would like to thank our Berkeley colleagues, namely Farokh Bastani, Abhijit Sahay, Andrew Guest, Atul Prakash, Jaideep Srivastava, Keshav, Moon Kim, and Yih-Farn Chen for their careful reviews and comments on various drafts of this article, which improved its readability a great deal. We are extremely grateful to the referees, who went through each and every line of the original draft to improve the presentation style and technical soundness.

References

1. L. Brownston et al., *Programming Expert Systems in OPS5: An Introduction to Rule-based Programming*, Addison-Wesley, Reading, MA, 1985.
2. "The Knowledge Engineering Environment," IntelliCorp, Knowledge Systems Div., Menlo Park, CA.

3. *KEE Software Development System User's Manual*, IntelliCorp Inc., Menlo Park, CA, 1985.
4. M. Stefik et al., "Knowledge Programming in LOOPS: Report on Experimental Course," *The AI Magazine*, Vol. 4, No. 3, Fall 1983.
5. D. G. Bobrow and M. Stefik, "The LOOPS Manual," Memo KB-VLSI-81-13, Xerox Corp., Palo Alto, CA, Aug. 1984.
6. *ART User's Manual*, Inference Systems Inc., CA, 1984.
7. A. S. Cromaty, "What Are Current Expert System Tools Missing?," *Proc. AAAI Conf.*, 1985.
8. S. J. Hansen and R. R. Rosinski, "Programmer Perception of Productivity and Programming Tools," *Comm. ACM*, Vol. 28, No. 2, Feb. 1985.
9. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Proc. IEEE 2nd Software Process Workshop*, 1986.
10. R. Davis et al., "The Dipmeter Advisor: Interpretation of Geological Signals," *Proc. 7th Int'l Joint Conf. AI*, Aug. 1981.
11. C. V. Ramamoorthy, V. Garg, and A. Prakash, "Programming in the Large," *IEEE Trans. Software Eng.*, Vol. SE-7, No. 7, July 1986.
12. A. N. Haberman et al., Special Issue on the Gerdorf Project, *J. Systems and Software*, Vol. 5, No. 2, May 1985.
13. C. V. Ramamoorthy et al., "Genesis—An Integrated Environment for Development and Evolution of Software," *COMPSAC*, 1985.
14. P. A. Kirlis et al., "The SAGA Approach to Large Program Development in an Integrated Modular Environment," *Proc. GTE Workshop on Software Engineering Environments for Programming-in-the-Large*, June 1985.
15. K. A. Frenkel, "Towards Automating the Software Development Cycle," *Comm. ACM*, Vol. 28, No. 6, June 1985.
16. T. Moto-oka, "Preliminary Report on Fifth Generation Computer Systems," keynote speech, *Proc. Int'l Conf. 5th Generation Computer Systems*, Tokyo, Japan, 1981.
17. M. Eisenstadt, "PTP: A Prolog Trace Package for the Prolog-10 Family," tech. report 12, Human Cognition Research Laboratory, The Open University, Milton Keynes, UK, 1984.
18. S. Hardy, "A New Software Environment for List Processing and Logic Programming," *Artificial Intelligence: Tools, Techniques, and Applications*, eds. Tom O'Shea and M. Eisenstadt, Harper and Row Publishers, New York, NY, 1984.
19. K. L. Clark, *Micro-Prolog*, Prentice Hall Inc., Englewood Cliffs, NJ, 1984.
20. P. Hammond, "APES (A Prolog Expert System Shell): A Users' Manual," doc. report 82/9, Dept. of Computer Science, Imperial College, London, UK, 1982.
21. Beau Sheil, "Power Tools for Programmers," *Datamation*, Feb. 1983.
22. W. Teitelman and L. Manister, "The Interlisp Programming Environment," *Computer*, Vol. 14, No. 4, Apr. 1981.
23. V. R. Basili and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Comm. ACM*, Jan. 1984.
24. R. C. Waters, "The Programmer's Apprentice: A Session with KBEmacs," *Trans. Software Eng.*, Vol. SE-11, No. 11, Nov. 1985.
25. G. Arrango et al., "Modeling Knowledge for Software Development," *IEEE 3rd Int'l Workshop on Software Specifications and Design*, 1985.
26. "Rome Air Development Center Report on Knowledge-Based Software Assistant," Kesterl Institute, CA, 1983.



C. V. Ramamoorthy is a professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He is editor-in-chief of *IEEE Trans. Software Engineering*.

Ramamoorthy received BS degrees in physics and technology from the University of Madras, Madras, India. He received the MS and the professional degree of mechanical engineer from UC Berkeley, and MA and PhD degrees in applied mathematics and computer theory from Harvard University.



Shashi Shekhar is a graduate student at the University of California, Berkeley, in the computer science division. His research interests include applied AI, software engineering, and distributed software systems.

Shashi received a BS in computer science from the Indian Institute of Technology, Kanpur, India.



Vijay Garg is a doctoral student at the University of California, Berkeley. His research interests include software engineering and distributed systems.

Garg received a BS in computer science from the Indian Institute of Technology, Kanpur, India, and an MS in computer science from UC Berkeley.

Readers may write to the authors at the Dept. of Electrical Engineering and Computer Sciences, Univ. of Calif., Berkeley, CA 94720.



Preliminary Call for Papers

Third Aerospace Computer Security Conference: Applying Technology to Systems

December 8-11, 1987

The Sheraton World Hotel
Orlando, Florida



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.



THE COMPUTER SOCIETY
OF THE IEEE

Sponsored by
The American Institute of Aeronautics and Astronautics
The American Society for Industrial Security
in cooperation with The IEEE Computer Society

The Conference

Operational requirements for civil and military systems under development increasingly stress the necessity for information to be readily accessible to users and operators. This produces an apparent conflict with policies and directives which require total protection of system data from compromises of privacy, confidentiality, and integrity. Accomplishing both of these sets of requirements requires the application of the maturing technology of computer security to new systems throughout their development cycle. In addition, operational approaches to satisfy system requirements and accommodate the implementation of engineering technology require intensified research and development.

This conference will explore technology applications in two complementary aspects: first, the policy issues and operational requirements for both civil and military systems; and second, the hardware and software tools and techniques being developed to satisfy system requirements. Special emphasis will be placed on specific examples of systems applications.

A three-day technical conference exploring the application of computer security technology will be preceded by a full day of tutorials dealing with policy matters, technology applications, and other areas. Full day introductory and advanced surveys will be offered as well as advanced courses exploring specialized technological areas.

Areas of Interest Include:

- Trusted DBMSs and Operating Systems
- Network Security
- Space Station Requirements
- Languages & Verification Tools
- Policy & Management Issues
- Advanced Architectures
- C³I Systems
- SDI Issues & Requirements
- Risk/Threat Assessments
- Data Privacy and Integrity

Papers, Panel Concepts and Tutorials

Technical papers, panel concepts, and tutorials which address the application of computer security technologies in the aerospace environment are solicited. Original research, analyses and approaches for defining the computer security issues and problems identified in the Conference's interest areas; methodological approaches for analyzing the scope and nature of computer security issues; and, potential solutions are of particular interest. Unclassified preliminary concepts/proposals should be submitted by **February 28, 1987**. Copies of final abstracts of approximately 1000 words must be mailed before **May 20, 1987**. Material should be sent to:

Papers and Panel Concepts

Mr. Stephen Walker,
Technical Program Chairman
Trusted Information Systems, Inc.
P.O. Box 45
Glenwood, MD 21738

Tutorials

Mr. J. Chris Perry
Tutorial Chairman
The Mitre Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102

Classified Sessions

Classified sessions at the secret level are planned. Preliminary unclassified abstracts and proposals for such sessions should be sent to the Technical Program Chairman.

Vendor Exhibit

A technical exhibit program is planned for interested vendors displaying computer and physical security products. Information regarding participation in this program will be mailed to vendors in early January, 1987. Inquiries at this time should be sent to the Conference General Chairman.

Addition Information

For more information or to receive future mailings, please contact Dr. Joel Levy, General Chairman at:

Aerospace Computer Security Associates
c/o ORI, Inc.
1375 Piccard Drive
Rockville, MD 20850

Don't Miss the Leading Applications— Oriented Forum for Users of AI Systems and Equipment

AI '87 emphasizes a widening array of applications for increasing productivity and reducing costs in business offices, industrial plants, laboratories and many professions.

Exhibition Highlights Newest Technology

Nearly 150 display booths of AI software and hardware provide a unique opportunity to evaluate



AI's potentials and to select from among the most current and sophisticated systems, equipment and technology available today.



3rd Annual Presentation

Artificial Intelligence and Advanced Computer Technology Conference/Exhibition

April 22-24, 1987

Long Beach Convention Center
Long Beach, California

Plan Now to Attend!

Mail Coupon Today for Details

TO: **Tower Conference Management Co.**
331 W. Wesley St., Wheaton, IL 60187

- ☐ I am interested in ATTENDING AI '87. Please send advance registration form and conference program details.
- ☐ My company is interested in EXHIBITING. Please send further information.

Name _____
Title _____
Company _____
Address _____
City _____ State/Country _____
Postal Code _____
Phone _____
Telex _____

AI '87 Conference Program to Feature Commercial, Industrial & Government Applications

More than 100 papers will be presented in 27 technical sessions, highlighting such topics as:

Expert Systems
AI Workstations
Manufacturing
AI Languages
Aerospace/Defense
Machine Translation
Medical/Health Care

Four Tutorials and a Workshop round out this informative, applications-oriented program.

Organized By:



**Tower Conference
Management Company**

331 W. Wesley St., Wheaton, IL 60187
(312) 668-8100 Telex: 350427

Co-Sponsored By:

DIGITAL DESIGN
THE ELECTRONIC SYSTEM DESIGN MAGAZINE

DM Data Inc.

SCS Society for
Computer Simulation

USAI

**Applied Artificial
Intelligence Reporter**

Symbolics Architecture

David A. Moon

Symbolics, Inc.

This architecture enables rapid development and efficient execution of large, ambitious applications. An unconventional design avoids trading off safety for speed.

What is an architecture? In computer systems, an architecture is a specification of an interface. To be dignified by the name architecture, an interface should be designed for a long lifespan and should connect system components maintained by different organizations. Often an architecture is part of a product definition and defines characteristics on which purchasers of that product rely, but this is not true of everything that is called an architecture. An architecture is more formal than an internal interface between closely-related system components, and has farther-reaching effects on system characteristics and performance.

A computer system typically contains many levels and types of architecture. This article discusses three architectures defined in Symbolics computers:

(1) System architecture—defines how the system appears to end users and application programmers, including the characteristics of languages, user interface, and operating system.

(2) Instruction architecture—defines the instruction set of the machine, the types of data that can be manipulated by those instructions, and the environment in which the instructions operate, for example subroutine calling discipline, virtual memory management, interrupts and exception traps, etc. This is an interface between the compilers and the hardware.

(3) Processor architecture—defines the overall structure of the implementation of the instruction architecture. This is an interface between the firmware and the hardware, and is also an interface between the parts of the processor hardware.

System architecture

System architecture defines how the system looks to the end user and to the programmer, including the characteristics of

languages, user interface, and operating system. System architecture defines the product that people actually use; the other levels of architecture define the mechanism underneath that implements it. System architecture is implemented by software; hardware only sets bounds on what is possible. System architecture defines the motivation for most of the design choices at the other levels of architecture. This section is an overview of Symbolics system architecture.

The Symbolics system presents itself to the user through a high-resolution bitmap display. In addition to text and graphics, the display contains presentations of objects. The user operates on the objects by manipulating the presentations with a mouse. The display includes a continuously updated reminder of the mouse commands applicable to the current context. Behind the display is a powerful symbol processor with specialized hardware and software. The system is dedicated to one user at a time and shares such resources as files, printers, and electronic mail with other Symbolics and non-Symbolics computers through both local-area and long-distance networks of several types. The local-area network is integral to system operation.

The system is designed for high-productivity software development both in symbolic languages, such as Common Lisp¹ and Prolog, and in nonsymbolic languages, such as Ada and Fortran. It is also designed for efficient execution of large programs, particularly in symbolic languages, and delivery of such programs to end users. The system is intended to be especially suited to complex, ambitious applications that go beyond what has been done before; thus it provides facilities for exploratory programming, complexity management, incremental construction of programs, and so forth. The operating system is written in Lisp and the architect-

tural concept originated at the MIT Artificial Intelligence Laboratory. However, applications are not limited to Lisp and AI. Many non-AI applications that are complex enough to be difficult on an ordinary computer have been successfully implemented.

Meeting these needs requires an extraordinary system architecture—just another PC or Unix clone won't do. The intended applications demand a lot of processor power, main and virtual memory size, and disk capacity. The system must provide as much performance as possible without exceeding practical limits on cost, and computing capacity must not be diluted by sharing it among multiple users. These purely hardware aspects are not sufficient, however. The system must also improve both the speed of software production and the quality of the resulting software by providing a more complete substrate on which to erect programs than has been customary. Programmers should not be handed just a language and an operating system and be forced to do everything else themselves.

At a high level, the Symbolics substrate provides many facilities that can be incorporated into user programs, such as user-interface management, embedded languages, object-oriented programming, and networking. At a low level, the substrate provides full run-time checking of data types, of array subscript bounds, of the number of arguments passed to a function, and of undefined functions and variables. Programs can be incrementally modified, even while they are running, and information needed for debugging is not lost by compilation. Thus the edit-compile-test program development cycle can be repeated very rapidly. Storage management, including reclamation of space occupied by objects that are no longer in use, is automatic so that the programmer does not have to worry about it; incremental so that it interferes minimally with response to the user; and efficient because it concentrates on ephemeral objects, which are the best candidates for reclamation. The system never compromises safety for the sake of speed. (A notorious exception, the dynamic rather than indefinite extent of *&rest* arguments, is recognized as a holdover from the past that is not consistent with the system architecture and will certainly be fixed in the future.)

In an ordinary architecture, such features would substantially diminish performance, requiring the introduction of switches to turn off the features and regain

speed. Our system architecture deems this unacceptable, because complex, ambitious application programs are typically never finished to the point where it is safe to declare them bug-free and remove run-time error-checking. We feel it is essential for such applications to be robust when delivered to end users, so that when something unanticipated by the programmer happens, the application will fail in an obvious, comprehensible, and controlled way, rather than just supplying the wrong answer. To support such applications, a system must provide speed and safety at the same time.

Symbolics systems use a combination of approaches to break the traditional dilemma in which a programmer must choose either speed or safety and comfortable software development:

- The hardware performs low-level checking in parallel with computation and memory access, so that this checking takes no extra time.
- Machine instructions are generic. For example, the Add instruction is capable of adding any two numbers regardless of their data types. Programs need not know ahead of time what type of numbers they will be adding, and they need no declarations to achieve efficiency when using only the fastest types of numbers. Automatic conversion between data types occurs when the two operands of Add are not of the same type.
- Function calling is very fast, yet does not lose information needed for debugging and does not prevent functions from being redefined.
- Built-in substrate facilities are already optimized and available for programmers to incorporate into their programs.
- Application-specific control of virtual-memory paging is possible. Pre-paging, postpurging, multipage transfers, and reordering of objects to improve locality are supported.²

These benefits are not without costs:

- Both the cost and the complexity of system hardware and software are increased by these additional facilities.
- Performance optimization is not always automatic. Programmers still must sometimes resort to metering tools. Declarations are available to optimize certain difficult cases, but their use is much less frequent than in conventional architectures.

Why Lisp machines? This is really three questions:

- (1) Why dedicate a computer to each user instead of time-sharing?

(2) Why use a symbolic system architecture?

(3) Why build a symbolic system architecture on unconventional lower-level architectures?

Why dedicate a computer to each user instead of time-sharing? This seemed like a big issue back in 1974 when Lisp machines were invented, but perhaps by now the battle has been won. A report from that era³ states these reasons for abandoning time-sharing:

- Time-sharing systems degrade under heavy load, so work on large, ambitious programs could only be conducted in off-peak hours. In contrast, a single-user system would perform consistently at any time of day.

- Performance was limited by the speed of the disk when running programs too large to fit in main memory. Dedicating a disk to each user would give better performance.

The underlying argument was that increasing program size and advancing technology, making capable processors much less expensive, had eliminated the economy of scale of time-sharing systems. The original purpose of time-sharing was to share expensive hardware that was only lightly used by any individual user. The serendipitous feature of time-sharing was interuser communication. Both of these purposes are now served by local-area networking. Expensive hardware units are still shared, but the processor is no longer among them.

These arguments apply to all types of dedicated single-user computers, even PCs, not only to symbolic architectures.

Why use a symbolic system architecture? Many users who need a platform for efficient execution of large symbolic programs, a high-productivity software development environment, or a system for exploratory programming and rapid prototyping have found symbolic languages such as Lisp and symbolic architectures such as this one very beneficial. Programs can be built more quickly, and fit more smoothly into an integrated environment, by incorporating such built-in substrate facilities as automatic storage management and the flexible display with its presentation-based user interface. The full error-checking saves time when developing new programs. The programmer can concentrate on the essential aspects of the program without fussing about minor

mistakes, because the machine will catch them. The ability to change the program incrementally greatly speeds up development.

Once the initial exploration phase is over, it is possible to turn prototypes into products quickly. Good performance can be achieved without a lot of programmer effort and without sacrificing those development-oriented features that are also of value later in the program's life, during maintenance and enhancement.

Why build a symbolic system architecture on unconventional lower-level architectures? Conventional instruction architectures are optimized to implement system architectures very different from Symbolics'. For example, they have no notions of parallel error-checking and generic instructions; they often obstruct the implementation of a fast function call, especially one that retains error-checking, incremental compilation, and debugging information; and they usually pay great attention to complex indexing and memory addressing modes, which have little utility for symbolic languages. Implementing Symbolics' system architecture on a conventional instruction architecture would force a choice between safety and performance: we could not have both. The type of software we are interested in either could not run at all or would require much faster hardware to achieve the same performance. Later I will discuss the special aspects of Symbolics' instruction and processor architectures that make them more suitable to support a symbolic system.

Comparing the performance of machines with equivalent cycle times and different architectures can sometimes be illuminating. The 3640, VAX 11/780, and 10-MHz 68020 all have cycle times of about 200 ns. (The 68020 takes two clock cycles to perform a basic operation, so its 100-ns nominal cycle time is equivalent to the other two machines' 200 ns.) On a Fortran benchmark (single-precision Whetstone), the VAX is 1.8 times the speed of the 3640 (750 versus 400). With floating-point accelerators on each machine, the ratio is 2.1. On the Lisp benchmark Boyer,⁴ the 3640 is 1.75 times the speed of the VAX running Portable Standard Lisp, 3.9 times the speed of the VAX running DEC Common Lisp, and 2.1 times the speed of the 68020 running Lucid Common Lisp. (The 68020 time at 10 MHz was estimated by multiplying its 16-MHz time by 1.6, no doubt an inaccurate procedure.) The VAX and 68020 programs were compiled with run-time error checking dis-

abled and safety compromised, while the 3640 was doing full checking as always: Like any benchmark figures presented without a complete explanation of what was measured, how it was measured, what full range of cases was tested, and how it can be reproduced in another laboratory, these numbers should not be taken very seriously. However, they give some idea of the effect of optimizing the instruction architecture to fit the system architecture. One could say that the VAX is three times better at Fortran than at Lisp and that the 68020 and VAX are similar for Lisp. These figures also show the effect of different compiler strategies on identical hardware.

This comparison was scaled to remove the effect of cycle time and show only the effect of architecture. This is not completely fair to the conventional machines, because in general they can be expected to have faster cycle times than a symbolic machine. Running the 68020 at full speed and using a newer model of the VAX would have improved their times. Hardware technology of conventional machines will always be a couple of years ahead of symbolic hardware, in cycle time and price/cycle, because of the driving force of their larger market. It's interesting to note that this hardware advantage applies only to the processor, which usually contributes less than 25 percent of system cost. Power supplies, sheet metal, and disk drives don't care whether the architecture is symbolic; they cost and perform the same for equivalent configurations of either type of machine.

This comparison is not completely fair to the symbolic machine, either. Software exploiting the full capabilities of the symbolic machine should have been compared, but this software won't run at all on the conventional machines. Software technology on symbolic machines will always be a couple of years ahead of conventional machines, because it is built on a more powerful substrate using more productive tools.

Performance. The best published analysis of performance of Lisp systems appears in Gabriel's work.⁴ The various 3600 models perform quite capably on these benchmarks, as can be seen from a perusal of the book. Some of the reasons for such good performance will become apparent as we proceed.

However, one must always ask exactly what a benchmark measures. A problem with Gabriel's benchmarks is that they are written in a least common denominator

dialect that represents Lisp as it was in 1970. This makes it easier to benchmark a broad spectrum of machines, but makes the benchmarks less valid predictors of the performance of real-world programs. Since 1970, there have been many advances in the understanding of symbolic processing and in the range of its applications. The basic operations measured by these benchmarks, such as function calling, small-integer arithmetic, and list processing, are still important today, but many other operations not measured are of equal importance. These benchmarks do not use the more modern features of Common Lisp (such as structures, sequences, and multiple values), do not use object-oriented programming, and are generally not affected by system-wide facilities such as paging and garbage collection. As predefined, portable programs, these benchmarks cannot benefit from the unusual aspects of Symbolics system architecture, such as large program support, full run-time safety, efficient storage management, substrate facilities, support for languages other than Lisp, and faster development of efficient programs.

Instruction architecture

Symbolics' philosophy is that different levels of architecture should be free to change independently, to satisfy different goals and constraints. Users see only the system architecture, leaving the lower levels, such as the instruction architecture, free to change to utilize available technology, maximize performance, or minimize cost. Most other computer families allow users to depend on the instruction architecture and therefore are not free to change it. It tends to be optimized for only the first member of the family. Later implementations using newer technology, as well as implementations at the high or low extremes of the price/performance curve, are penalized by the need for compatibility with an unsuitable instruction architecture.

Symbolics system architecture has been implemented on three different instruction architectures. The LM-2 machine, based on the original MIT Lisp Machine,³ was the first; it was discontinued in 1983. The 3600 family of machines uses a second instruction architecture and three different processor architectures. A third instruction architecture, appropriate for VLSI implementation, is being used in a line of future products now under development.

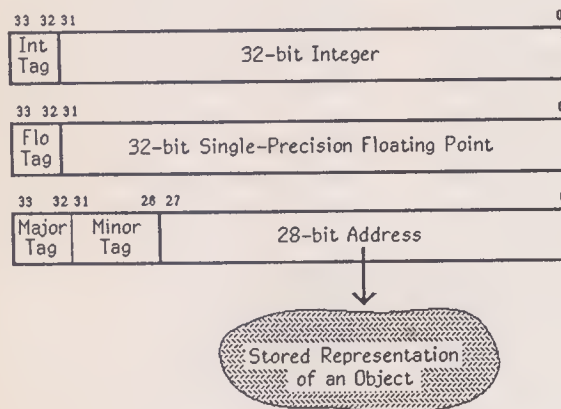


Figure 1. An object reference is a 34-bit quantity, consisting either of a 32-bit data word with a 2-bit data type tag, or of a 28-bit address with a 6-bit data type tag.

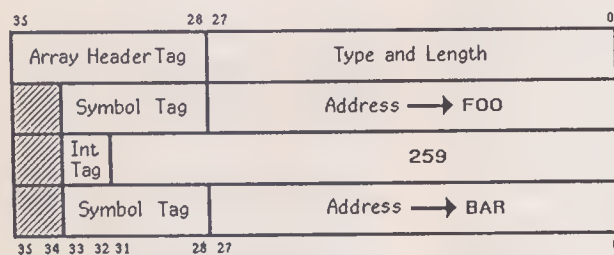


Figure 2. An array of three elements—FOO, 259, and BAR—consists of a header word defining the type and length of the array, followed by an object reference for each array element.

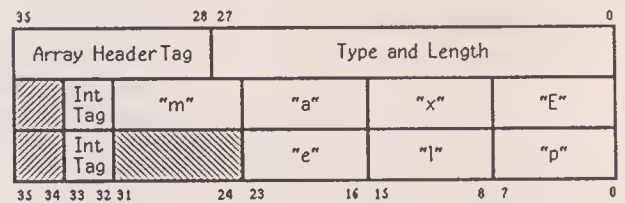


Figure 3. A string containing the seven characters "Example" stores each character in a single 8-bit byte. Bytes are packed into 32-bit integer objects.

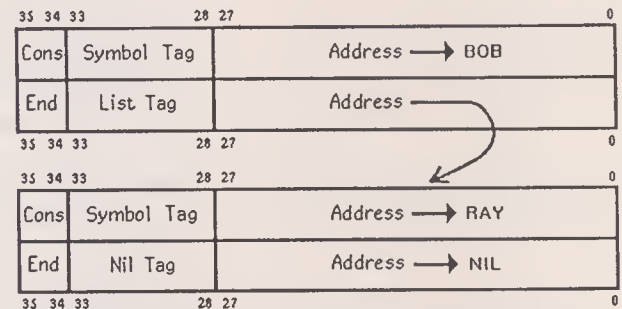


Figure 4. An ordinary list of two elements requires four words of storage. Unlike arrays, lists do not have headers.

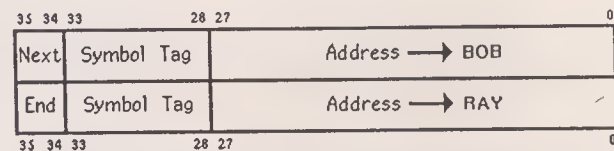


Figure 5. A compact list of two elements requires two words of storage. It uses the cdr code to eliminate two object references.

(Reprinted from "Architecture of the Symbolics 3600," 12th Int'l Symp. Computer Architecture, © 1985 IEEE.)

The following sections summarize the instruction and processor architectures of the 3600 family, discuss some of the design tradeoffs involved, and show how these architectures are especially effective at supporting the desired system architecture. Further details can be found elsewhere.^{5,6}

Data are object references. The fundamental form of data manipulated by any Lisp system is an *object reference*, which designates a conceptual object. The values of variables, the arguments to functions, the results of functions, and the elements of lists are all object references. There can be more than one reference to a given object. Copying an object reference makes a new reference to the same object; it does not make a copy of the object.

Variables in Lisp and variables in conventional languages are fundamentally different. In Lisp, the value of a variable is an object reference, which can refer to an object of any type. Variables do not intrinsically have types; the type of the object is encoded in the object reference. In a conventional language, assigning the value of one variable to another copies the object, possibly converts its type, and loses its identity.

A typical object reference contains the address of the object's representation in storage. There can be several object references to a particular object, but it has only a single stored representation. Side-effects to an object, such as changing the contents of one element of an array, are implemented by modifying the stored represen-

tation. All object references address the same stored representation, so they all see the side-effect.

In addition to such object references by address, it is possible to have an immediate object reference, which directly contains the entire representation of the object. The advantage is that no memory needs to be allocated when creating such an object. The disadvantage is that copying an immediate object reference effectively copies the object. Thus, immediate object references can only be used for object types that are not subject to meaningful side-effects, have a small representation, and need very efficient allocation of new objects. Small integers (traditionally called fixnums) and single-precision floating-point numbers are examples of such types.

In the 3600 architecture, an object reference is a 34-bit quantity consisting of a 32-bit data word and a 2-bit major data type tag. The tag determines the interpretation of the data word. Often the data word is broken down into a 4-bit minor data type tag and a 28-bit address (see Figure 1). This variable-length tagging scheme accommodates industry-standard 32-bit fixed and floating-point numbers with a minimum of overhead bits for tagging. Addresses are narrower than numbers to make additional tag bits available for the many types of objects that Lisp uses.

Addresses are 28 bits wide and designate 36-bit words in a virtual memory with 256-word pages. The address granularity is a word, rather than a byte as in many other machines, because the architecture is object-oriented and objects are always aligned on a word boundary. This results in one gigabyte of usable virtual memory. It is interesting to note that the 3600's 28-bit address can actually access the same number of usable words as the VAX's 32-bit address, because the VAX expends two bits on byte addressing and reserves three-fourths of the remaining address space for the operating system kernel and the stack (neither of which is large).

In addition to immediate and by-address object references, the 3600 also uses pointers, a special kind of object reference that does not designate an object as such. A pointer designates a particular location within an object or a particular instruction within a compiled function. Pointers are used primarily for system programming.⁷

Stored representations of objects. The stored representation of an object is contained in some number of consecutive words of memory. Each word may contain an object reference, a header, a special marker, or a forwarding pointer. The data type tags distinguish these types of words. For example, an array is represented as a header word, containing such information as the length of the array, followed by one memory word for each element of the array, containing an object reference to the contents of that element (see Figure 2). An object reference to the array contains the address of the first memory word in the stored representation of the array.

A *header* is the first word in the stored representation of most objects. A header marks the boundary between the stored representations of two objects. It contains descriptive information about the object that it heads, which can be expressed as

either immediate data or an address, as in an object reference.

A *special marker* indicates that the memory location containing it does not currently contain an object reference. Any attempt to read that location signals an error. The address field of a special marker specifies what kind of error should be signalled. For example, the value cell of an uninitialized variable contains a special marker that addresses the name of the variable. An attempt to use the value of a variable that has no value provokes an error message that includes the variable's name.

A *forwarding pointer* specifies that any reference to the location containing it should be redirected to another memory location, just as in postal forwarding. These are used for a number of internal bookkeeping purposes by the storage management software, including the implementation of extensible arrays.

Some objects include packed data in their stored representation. For example, character strings store each character in a single 8-bit byte (see Figure 3). For uniformity, the stored representation of an object containing packed data remains a sequence of object references. Each word is an immediate object reference to an integer, whose 32 bits are broken down into packed fields as required, such as four 8-bit bytes in the case of a character string.

A word in memory consists of 36 bits, of which I have already explained 34. When a memory word contains a header or a machine instruction, the remaining two bits serve as an extension of the rest of the word. When a memory word contains an object reference, a special marker, or a forwarding pointer, the remaining two bits are called the *cdr* code. The representation of conses and lists (Steele, p. 26)¹ saves one word by using the *cdr* code instead of a separate header to delimit the boundaries of these small objects. In addition, lists are represented compactly by encoding common values of the *cdr* in the *cdr* code instead of using an object reference (see Figures 4 and 5).

Tagging every word in memory produces these benefits:

- All data are self-describing and the information needed for full run-time checking of data types, array subscript bounds, and undefined functions and variables is always available.
- Hardware can process the tag in parallel with other hardware that processes the rest of a word. This makes it possible to optimize safety and speed simultaneously.

- Generic instructions alter their operation according to the tags of their operands.

- Automatic storage management is simple, efficient, and reliable. It can be assisted by hardware, since the data structures it deals with are simple and independent of context. The details appear elsewhere.^{8,5}

- Data use less storage due to compact representations. Programs use less storage due to generic instructions and because tag checking is done in hardware, not software.

The cost of tagging is that more main memory and disk space are required to store numerical information. Each main memory word includes 7 bits for error detection and correction, so the 4 tag bits add 10 percent. Each 256-word disk sector includes about 128 bytes of formatting overhead, so the 4 tag bits per word add 11 percent. We feel that the benefits amply justify these costs.

Instruction set. The 3600 architecture includes an instruction set produced by the compilers and executed by a combination of hardware and firmware. All instructions are 17 bits long, consisting of a 9-bit operation field and an 8-bit argument field. Instructions are packed two per word, which is important for performance in two ways:

- (1) Dense code decreases paging overhead by making programs occupy fewer pages and

- (2) simplifies the memory system by decreasing the ratio of required instruction fetch bandwidth (in words/second) to processor speed (in instructions/second).

Every instruction is contained in a compiled function, which consists of some fixed overhead, a table of constants, and a sequence of instructions (see Figure 6). The table of constants contains object references to objects used by the instructions, including locative pointers to definition cells of functions called by this function. Indirection through the definition cell ensures that if a function is redefined its callers are automatically linked to the new definition.

Instructions operate in a stack machine model: Many instructions pop their operands off the stack and push their results onto the stack. In addition to these 0-address instructions, there are 1-address instructions, which can address any location in the current stack frame. In this way the slots of the current stack frame serve the same purpose as registers. The

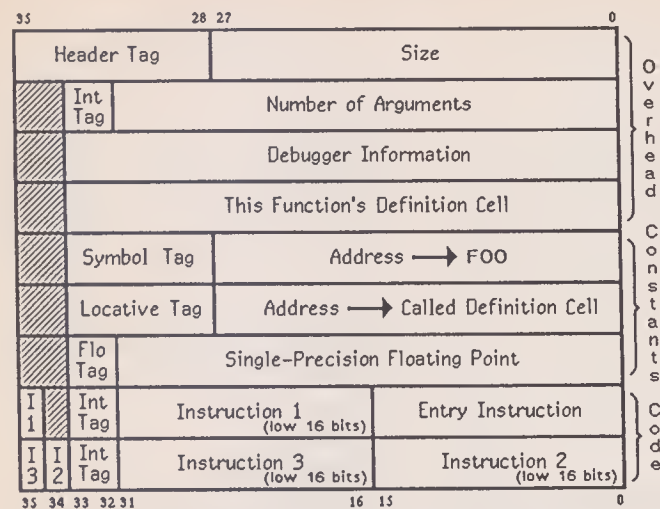


Figure 6. A compiled function consists of four words of overhead, a table of constants and external references, and a sequence of 17-bit instructions, packed two per word.

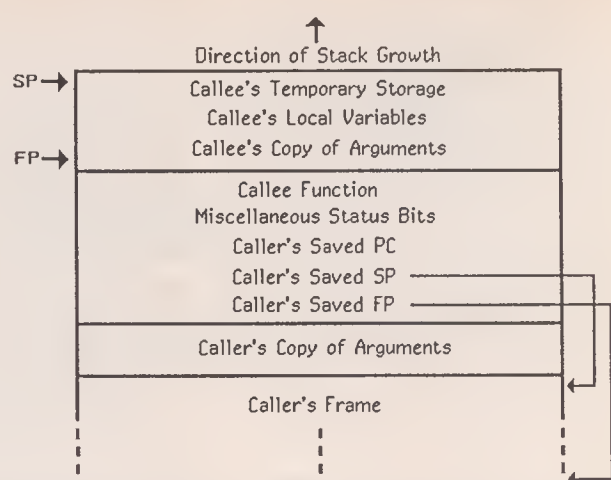


Figure 7. A stack frame consists of the caller's copy of the arguments, five header words, the callee's copy of the arguments, local variables, and temporary storage. The frame-pointer (FP) and stack-pointer (SP) registers address the current stack frame.

(Reprinted from "Architecture of the Symbolics 3600," 12th Int'l Symp. Computer Architecture, © 1985 IEEE.)

1-address instructions include multi-operand instructions, which pop all of their operands except the last off the stack and take their last operand from a location in the current stack frame.

There are several ways an instruction can use its argument field. Table 1 lists the ways to develop the address of an operand in the stack or in memory by adding argument to a base address. Table 2 lists non-address uses of argument. Each individual opcode only uses argument in a single way; there are no addressing modes. The motivation for implementing this particular set of arguments is to provide for constants (including small integers as a special case), all types of Lisp variables (local and nonlocal lexical, special, structure slot, instance), branching, and byte fields. Byte fields were included because they are heavily used in system programming.

Many instructions are simply Lisp functions directly implemented by hardware and firmware, rather than built up from other Lisp functions and implemented as compiled instructions. These Lisp-function instructions are known as built-ins. They take a fixed number of arguments from the stack and from their argument field. They return a fixed number of values on the stack. Examples of built-ins are eq, symbolp, logand (with two arguments), car, cons, member, and aref (with two arguments).¹ The criterion for implementing a Lisp function as a built-in in-

struction is that hardware is only used to optimize key performance areas. When a Lisp function is not critical to system performance, or hardware implementation of it cannot achieve a major speedup, it remains in software where it is easier to change, to debug, and to optimize.

Using an instruction set designed for Lisp rather than adapting one designed for Fortran or for a hand-crafted assembly language enhances safety and speed. 3600 instructions always check for errors and exceptions, so programs need not execute extra instructions to do that checking. Instructions operate on tagged data, so extra instructions to insert and remove tags are not needed. Instructions are generic, so declarations are not needed to tell the compiler how to select type-specific instructions and translate between data formats. In contrast, Lisp compilers for conventional machines⁹ must generate extra shifting or masking instructions to manipulate tags, must use multi-instruction sequences for simple arithmetic operations unless there are declarations, and are always having to compromise between safety and speed.

Unlike many machines, the 3600 does not have indexed and indirect addressing modes. Instead it has instructions that perform structured, object-oriented operations such as subscripting an array or fetching the car of a list. This fits the instruction set more closely to the needs of

Lisp and at the same time simplifies the hardware by reducing the number of instruction formats to be decoded.

Function call. Storage whose lifetime is known to end when a function returns (or is exited abnormally) is allocated in three stacks, rather than in the main object storage heap, to increase efficiency. The control stack contains function-nesting information, arguments, local variables, function return values, and small stack-allocated temporary objects. The binding stack records dynamically bound variables.¹ The data stack contains stack-allocated temporary objects. This article concentrates on the control stack, which is the most critical to performance.

The protocol for calling a function is to push the arguments onto the stack, then execute a Call instruction that specifies the function to be called, the number of arguments, and what to do with the values returned by the function. When the function returns, the arguments have been popped off the stack and the values (if wanted) have been pushed on. Note the similarity in interface between functions and built-in instructions.

Every time a function is called, a new stack frame is built on the control stack. A stack frame consists of the caller's copy of the arguments, five header words, the callee's copy of the arguments, local variables, and temporary storage, including

arguments being prepared for calling the next function (see Figure 7). The current stack frame is delimited by the frame-pointer (FP) and stack-pointer (SP) registers, which are available as base registers in instructions that use their argument field to address locations in the current stack frame.

A compiled function starts with a sequence of one or more instructions known as the entry vector. The first instruction in the entry vector, the entry instruction, describes how many arguments the function accepts, the layout of the entry vector, and the size of the function's constants table (see Figure 6), and tells the Call instruction where in the entry vector to transfer control. The Call instruction and the entry vector cooperate to copy the arguments to the top of the stack (creating the callee's copy), convert their arrangement in storage if required, supply default values for optional arguments that the caller does not pass, handle the &rest and Apply features of Common Lisp, and signal an error if too many or too few arguments were supplied. The details are beyond the scope of this article.

Function return. A function returns by executing a Return instruction whose operands are the values to be returned. The value disposition saved in the frame header by Call controls whether Return discards the values, returns one value on the stack, returns multiple values with a count on the stack, or returns all the values to the caller's caller.

Return removes the current frame from the stack and makes the caller's frame current, by restoring the saved FP, SP, and PC registers. If the cleanup bits in the frame header are nonzero, special action must be taken before the frame can be removed. Return takes this action, clears the bit, and tries again. Cleanup bits are used to pop corresponding frames from the binding and data stacks, for unwind-protect,¹ for debugging and metering purposes, and for stack buffer housekeeping.

Motivations of the function call discipline. The motivations for this particular function-calling discipline are

- to implement full Common Lisp function calling efficiently,
- to be fast, so that programmers will write clear programs,
- to retain complete information for the Debugger, and
- to be simple for the compiler.

Table 1. Ways to develop an operand address.

Beginning of the current stack frame + offset
 End of the current stack frame - offset
 Lexical parent's environment captured in a closure + offset
 Current function's constants table + offset (possibly with indirection)
 An array used as a DEFSTRUCT structure + subscript
 A flavor instance + offset (possibly with mapping-table indirection)
 Address of the current instruction $\pm$ offset (for branching)

Table 2. Nonaddress uses of argument field.

A byte field any number of bits wide, positioned anywhere within a 32-bit word. This specification takes ten bits and hence overflows into two bits of operation.
 An immediate integer between -128 and 127
 An immediate integer between 0 and 255
 Extended operation field, to allow more than 512 instructions

To implement full Common Lisp function calling efficiently requires matching the arguments supplied by the caller—with normal function calling or with Apply—o the normal, &optional, and &rest parameters of the callee, and generating default values for unsupplied optional arguments. The entry vector takes care of this. Common Lisp's &key parameters are implemented by accepting an &rest parameter containing the keywords and values, then searching that list for each &key parameter. Multiple values are passed back to the caller on the stack, with a count. The caller reconciles the number of values returned with the number of values desired.

Function calling historically has been a major bottleneck in Lisp implementations, both on stock hardware and on specially-designed Lisp machines. It is important for function calling to be as fast as possible. If it is not, efficiency-minded programmers will distort their programming styles to avoid function calling, producing code that is hard to maintain, and will waste a lot of time doing optimization by hand that should have been done by the Lisp implementation itself. The 3600's function call mechanism attains good speed (fewer than 20 clock cycles for a one-argument function call and return when no exceptions occur) by using a stack buffer to minimize the number of memory references required, by optimizing the stack frame layout to maximize speed rather than to minimize space, by arranging for the checks for slower exception

cases to be fast (for example, Return simply checks whether the cleanup bits are nonzero), and by using the entry vector mechanism to simplify run-time decision-making.

The information that the debugger can extract from a stack frame includes the address of the previous frame (from the saved FP in the header), the function running in that frame (from the header), the current instruction in that function (from the PC saved in the next frame), the arguments (from the stack—the header specifies the argument count and arrangement), the local variables (from the stack), and the names of the arguments and local variables (from a table created by the compiler and attached to the function).

The compiler is simple because there is only a single calling sequence. Any call can call any function, and the argument patterns are matched up at run time. Everything is in the stack and no register-saving conventions are required, since there are no general-purpose registers.

The principal costs of this function-calling discipline are the five-word header in each frame and the copying of arguments to the top of the stack. The time to create the header is not a problem, because it is overlapped with necessary memory accesses, but the space occupied by the header and by the extra copy of the arguments is a substantial fraction of the typical frame size. This extra space is not a major problem because the stack buffer is large enough (1024 words) that it rarely overflows.

Argument copying is necessary because Common Lisp functions do not take a fixed number of arguments. In a function with &optional parameters, some of the arguments are supplied by the caller while the others are defaulted by the entry vector. The location in the stack frame of an argument must not depend on whether it was supplied or defaulted, since this varies from one call to the next, but the compiler must know the location in order to generate code to access the argument. The entry vector could not put default values in the standard location if the arguments were not at the top of the stack, because the frame header would be in the way. In a function with an &rest parameter, the caller can supply an arbitrary number of arguments. If these arguments were at the top of the stack, they would make it impossible for the compiler to know the locations of the local variables, which are pushed after the arguments.

Copying the arguments that are not part of an &rest parameter to the top of the stack solves both these problems. It gives the function complete control over the arrangement of its stack frame and makes the stack depth constant. Argument copying takes extra time, but typically only one clock cycle per argument, which is faster than the run-time decision-making that would otherwise be necessary to access an optional argument or a local variable.

Processor architecture

Three processor architectures are used in three representative models of the 3600 family: 3640, 3675, and 3620. Since they all implement the same instruction architecture, there are substantial similarities among their processor architectures. They differ due to implementation in different technologies and choices of different cost/performance tradeoffs, but this overview largely glosses over the differences.

The main goal of each of these processor architectures is to implement the instruction architecture described earlier with the highest performance achievable within its particular cost budget. The costs are generally higher than most workstations but lower than most minicomputers. For high performance the number of clock cycles required to execute an instruction must be minimized; the goal is to execute a new instruction every cycle. Because the system architecture specifies that safety and convenience must not be compromised to increase performance, instruc-

tions typically make many checks for errors and exceptions. Minimizing the cycle count demands that these checks be performed in parallel, not each in a separate cycle.

Adequate bandwidth for access to operands is also required. In the 3600 instruction architecture, a simple instruction can read two stack locations and write one stack location. One of these is a location in the current stack frame specified by an address in the instruction, while the other two are at the top of the stack. Operands are supplied by the stack buffer, a 1K-word memory that holds up to four virtual-memory pages of the stack. The stack buffer contains all of the current frame plus as many older frames as happen to fit. When the stack buffer fills up (during Call), the oldest page spills into normal memory to make room for the new frame. When the stack buffer becomes empty (during Return), pages move from normal memory back into the stack buffer until the frame being returned to is entirely in the buffer. The maximum size of a stack frame is limited to what will fit in the stack buffer. A second stack buffer contains an auxiliary stack for servicing page faults and interrupts without disturbing the primary buffer.

Associated with the stack buffer are the FP and SP registers, which point to the current frame and to the top of the stack, and hardware for addressing locations in the current stack frame via the argument field of an instruction, which calculates a read address and a write address every clock cycle. The third operand access is provided by a duplicate copy of the top location in the stack, in a scratchpad memory, which can be read and written every clock cycle. The SP register is incremented or decremented by instructions that push or pop the stack.

The stack buffer provides the same operand bandwidth, two reads and one write every clock cycle, as in a typical register-oriented architecture. It has the advantage that register saving and restoring across subroutine calls is not required, since all registers already reside in the stack. As in a register-window design, overhead occurs only when the stack buffer overflows or underflows and requires a block transfer between stack buffer and main memory. Another advantage is that each instruction contains only one address instead of three, making the instructions smaller (so that they can be fetched from main memory more quickly and processed with less hardware) and allowing more

registers to be addressed. A disadvantage of a stack architecture is that it requires address-calculation hardware, including a 10-bit (for a 1K-word buffer) adder. Since each instruction contains only one address instead of three, extra instructions are sometimes required to move data to the top of the stack so they can be addressed.

Instructions are processed by a four-stage pipeline (see Figure 8) under the control of horizontal microcode. Microcode is used as an engineering technique, not to create a general-purpose emulator that could implement alternate instruction architectures. Knowledge of the instruction architecture is built into hardware wherever that achieves a substantial performance improvement.

To achieve full performance, instructions must be supplied to the processor at an adequate rate. Each processor model has a different design, with different tradeoffs.

The 3640 uses a four-instruction buffer. When the buffer is exhausted, or a branch occurs, microcode reads two words from memory and refills the instruction buffer. This design uses much less hardware than the other two, but provides lower performance. Refilling the buffer takes five clock cycles, so in the worst case the performance penalty is about a factor of two. With a typical instruction mix, the observed slowdown is about 35 percent, because complex instructions such as function calls and memory references spend more than one cycle in the execute stage.

The 3675 uses a 2K-instruction cache. Program loops that fit in the cache execute at full speed, with no instruction fetching overhead. An autonomous instruction prefetch unit fills the cache with instructions before they are needed, in parallel with execution. At the cost of a substantial increase in hardware complexity over the 3640, this design ensures that the pipeline almost never has to wait for an instruction.

The 3620 uses a six-instruction buffer. An autonomous instruction prefetch unit fills the buffer in parallel with execution. The 3620 instruction stage is a compromise between the other two designs. Straight-line code executes at full speed, but branches execute at 3640 speed because they must refill the buffer.

The datapath contains several units that function in parallel (see Figure 9). Simple instructions such as data movement, arithmetic, logical, and byte-field instructions execute in a single clock cycle. For example, when executing an Add instruction

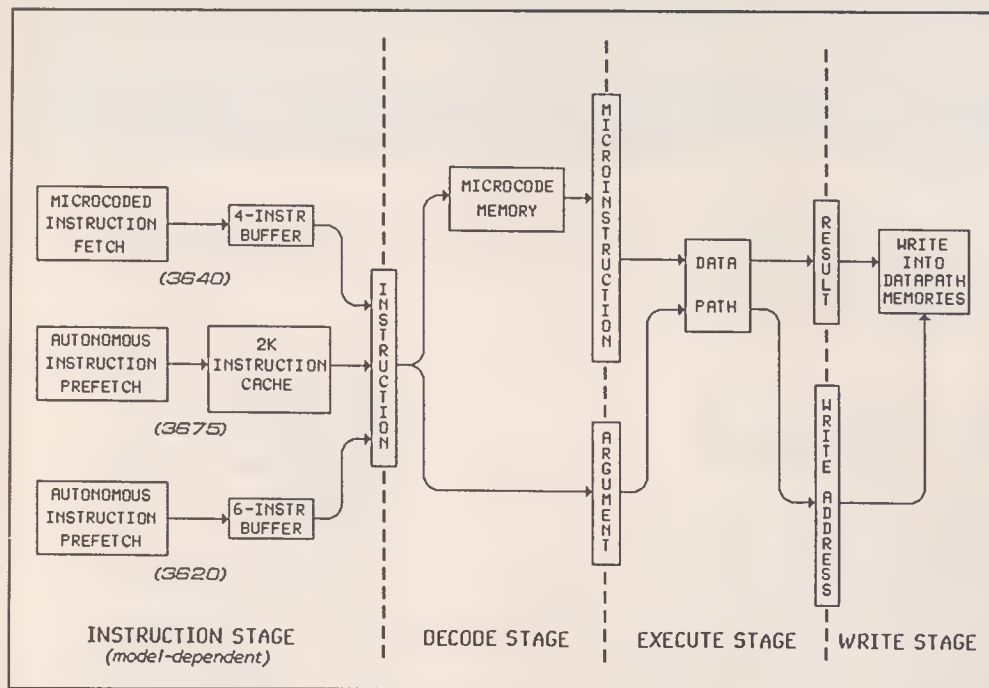


Figure 8. The instruction processing pipeline, with variations for three 3600 family models.

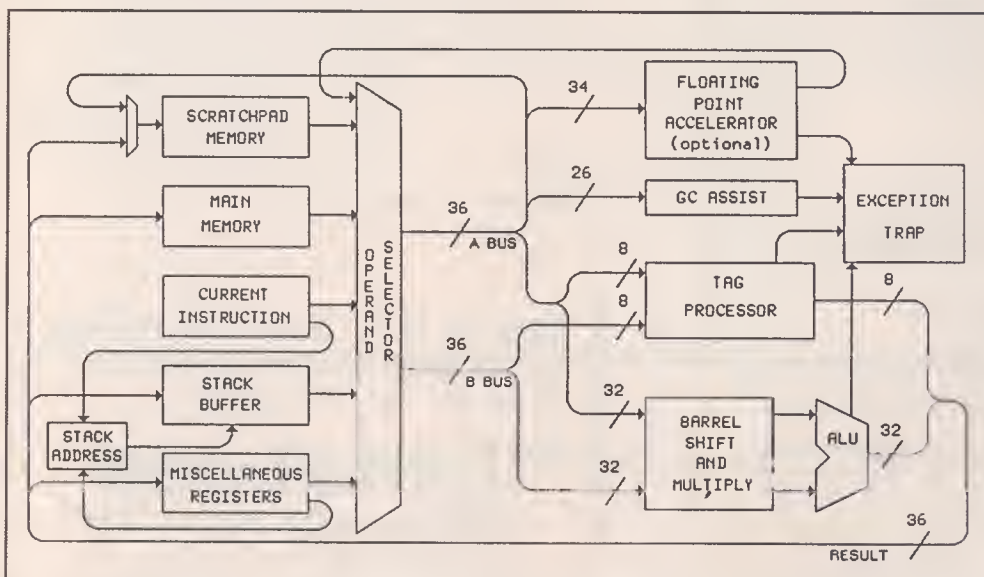


Figure 9. 3640 datapath, contained in the Execute and Write stages of the pipeline. Other 3600 family models have generally similar datapaths.

the following activities all take place in parallel:

- The stack buffer fetches the two operands, one from a calculated address in the stack buffer memory and the other from the duplicate top-of-stack in the scratchpad memory.
- The fixed-point arithmetic unit computes the 32-bit sum of the operands and checks for overflow. This result is only used if both operands are fixnums.
- The optional floating-point accelerator, if present, starts computing the sum of the operands and checking for floating-point exceptions. This result is

only used if both operands are single-floats.

- The tag processor checks the data types of the operands.
- The stack buffer accepts the result from the fixed-point arithmetic unit, adjusts the stack pointer, and in the write stage stores the result at the new top of the stack.
- The decode stage decodes the next instruction and produces the microinstruction that will control its execution. If the type-checking unit or either arithmetic unit detects an exception, control is diverted to a microcode exception handler.

When the operands of Add are not both fixnums, executing the instruction takes more than one machine cycle and more than one microinstruction. In the case of adding two single-floats, the extra time is only required because the floating-point arithmetic unit is slower than the fixed-point arithmetic unit. In other cases, extra time is required to convert the operands to a common format, to perform double-precision floating-point operations, or to trap to a Lisp function to add numbers of less common types.

Memory-reference instructions such as the car and aref Lisp operations are limited

mainly by the speed of the memory. Car, for example, takes four clock cycles. Complex instructions such as Call, Return, and the Common Lisp member function invoke microcode subroutines. A wide microinstruction word and fast microcode branching minimize the number of microinstructions that need to be executed. Simple and memory-reference instructions can be discovered to be complex at run time because of an exceptional condition such as the data type of the operands.

I have described here an unusual system architecture and presented an overview of the underlying architectures that implement it. When considering the type of applications that this system architecture targets, note how important to their success it is that we compromise neither safety nor speed. With this in mind, some of the unconventional design choices in these architectures were made based on rationales with varied benefits and costs. For example, a close fit between processor, instruction, and system architectures improves performance, but allowing users to depend on details of the instruction architecture can interfere with

this. The lack of this close fit dissipates the hardware price/performance advantage of conventional architectures when measuring system-level performance on software suited to symbolic architectures. □

References

1. G. L. Steele, *Common Lisp*, Digital Press, Burlington, MA, 1984.
2. D. L. Andre, *Paging in Lisp Programs*, Master's thesis, University of Maryland, 1986.
3. R. D. Greenblatt et al., "The LISP Machine," *Interactive Programming Environments*, eds. D. R. Barstow, H. E. Shrobe, and E. Sandewall, McGraw-Hill, Hightstown, NJ, 1984.
4. R. P. Gabriel, *Performance and Evaluation of Lisp Systems*, The MIT Press, Cambridge, MA, 1985.
5. D. A. Moon, "Architecture of the Symbolics 3600," *12th Int'l Symp. Computer Architecture*, 1985, pp. 76-83.
6. *Symbolics Technical Summary*, Symbolics Inc, Cambridge, MA, 1985.
7. *Symbolics Common Lisp: Language Dictionary*, Symbolics Inc, Cambridge, MA, 1986.

8. D. A. Moon, "Garbage Collection in a Large Lisp System," *Proc. 1984 ACM Symp. Lisp and Functional Programming*, pp. 235-246.
9. R. A. Brooks et al., "Design of an Optimizing, Dynamically Retargetable Compiler for Common Lisp," *Proc. 1986 ACM Conf. Lisp and Functional Programming*, pp. 67-85.



David A. Moon is a technical director at Symbolics, Inc. Previously, he was a hardware designer, microprogrammer, and writer of manuals at Symbolics. His interests include advanced software development and architectures for symbolic processing.

Moon received the BS degree in mathematics from MIT in 1975.

Readers may write to the author at Symbolics, Inc., 11 Cambridge Center, Cambridge, MA 02142. His e-mail address is Moon@Stony-Brook. SCRC.Symbolics.COM on the ARPA Internet.

RCI IS REACHING NEW PLATEAUS IN PARAMETRIC SOFTWARE
COST ESTIMATING MODELS WITH

SOFTCOST-R

Utilizing the Revolutionary Efforts of Dr. R. Teusworthe et the renowned Jet Propulsion Laboratory, RCI has developed a Cost Estimating Package that encompasses the requirements that up until now were just good ideas. Only SOFTCOST-R can provide you with features that include:

- "What-If" Capacity that Enables Rapid Analysis of your project's Cost and Schedules
- A Work Breakdown Structure that lets you Tie-In with Automatic Gantt Schedule and Pert Chart Generation
- A way to bound Risk by Computing the Confidence of your estimate of Time, Effort and Size through a Series of Submodels
- Ease of Use and Understanding as well as Support Services with available Training, User's Group Annual Conference, Quarterly User's Newsletter, Maintenance, Consultations, and now a Tutorial is in the process of being developed.
- An ability to run on IBM PC and Compatibles, and to be Calibrated to your Specific Environment
- Generation of Many Useful Reports for Managers and Cost Analysts such as: Resources Reports, Input Value Summary Reports, Project Estimate Summary Reports and others
- Uses the 1986 version of the popular COCOMO model as a sanity check
- Lets you evaluate the implications of ADAtm and incremental development on your workforce allocations decisions

Now is the time to become one of the Many Successful Organizations who have acquired the benefits of SOFTCOST-R by making it their primary Software Cost Estimating Package. For further information, write or call today:



Reiter Consultants, Inc.

25550 Hawthorne Boulevard, Suite 208
Torrance, California 90505(213) 373-8728

New in 1987 from Macmillan:

A practical new textbook on database design and data management!

The Database Book

MARY E. S. LOOMIS
465 pages

- Emphasizes the practical application of principles and the importance of design in database development.
- Mathematical treatment of concepts has been kept to a minimum—orientation is toward practical applications for both business and engineering.
- Covers the 3-scheme approach for implementing and controlling distributed databases.
- Features chapters on: Logical data modeling techniques, logical design of network databases, and data dictionaries.
- Each chapter concludes with discussion questions, problems, and exercises.
- Techniques presented throughout text will enable students to work successfully with any commercial/research database management system.

Look to Macmillan for your textbook needs. Call Toll-Free 1-800-428-3750, or write:

Macmillan Publishing Company
College Division/866 Third Ave./New York, NY 10022

THE MOVING FRONTIER . . . Spring '87

AI in the 1980s and Beyond

An MIT Survey

*edited by Eric L. Grimson and
Ramesh S. Patil*

May \$24.95

A Few Good Men from UNIVAC

David E. Lundstrom

June \$19.95

Image Synthesis

Elementary Algorithms

Gerard Hégon

May \$20.00

An Artificial Intelligence Approach to Legal Reasoning

Anne von der Lieth Gardner

April \$22.50

Advanced Research in VLSI

Proceedings of the 1987 Stanford Conference

edited by Paul Losleben

July \$40.00

Principles of Operating Systems

Sacha Krakowiak

May \$37.50

Digital Halftoning

Robert Ulichney

April \$35.00

The Characteristics of Parallel Algorithms

*edited by Leah H. Jamieson, Dennis B.
Gannon, and Robert J. Douglass*

May \$27.50

Microprogrammable Parallel Computer

MUNAP and its Applications

Takanobu Baba

June \$30.00

Theory of Deductive Systems and Its Applications

S. Yu. Maslov

translated by Michael Gelfond and Vladimir Lifschitz

July \$20.00

New Paperbacks

Theory of Recursive Functions and Effective Computability

Hartley Rogers, Jr.

April \$15.00 paperback

LOGO

Exploring Language with Logo Computer Science Logo Style

E. Paul Goldenberg and Wallace Feurzeig

June \$19.95 paperback

Volume 1. Intermediate Programming

Volume 2. Projects, Styles, and Techniques

Volume 3. Advanced Topics (forthcoming, June)

Brian Harvey

\$19.95 each, paperback

Visual Modeling with Logo

A Structured Approach to Seeing

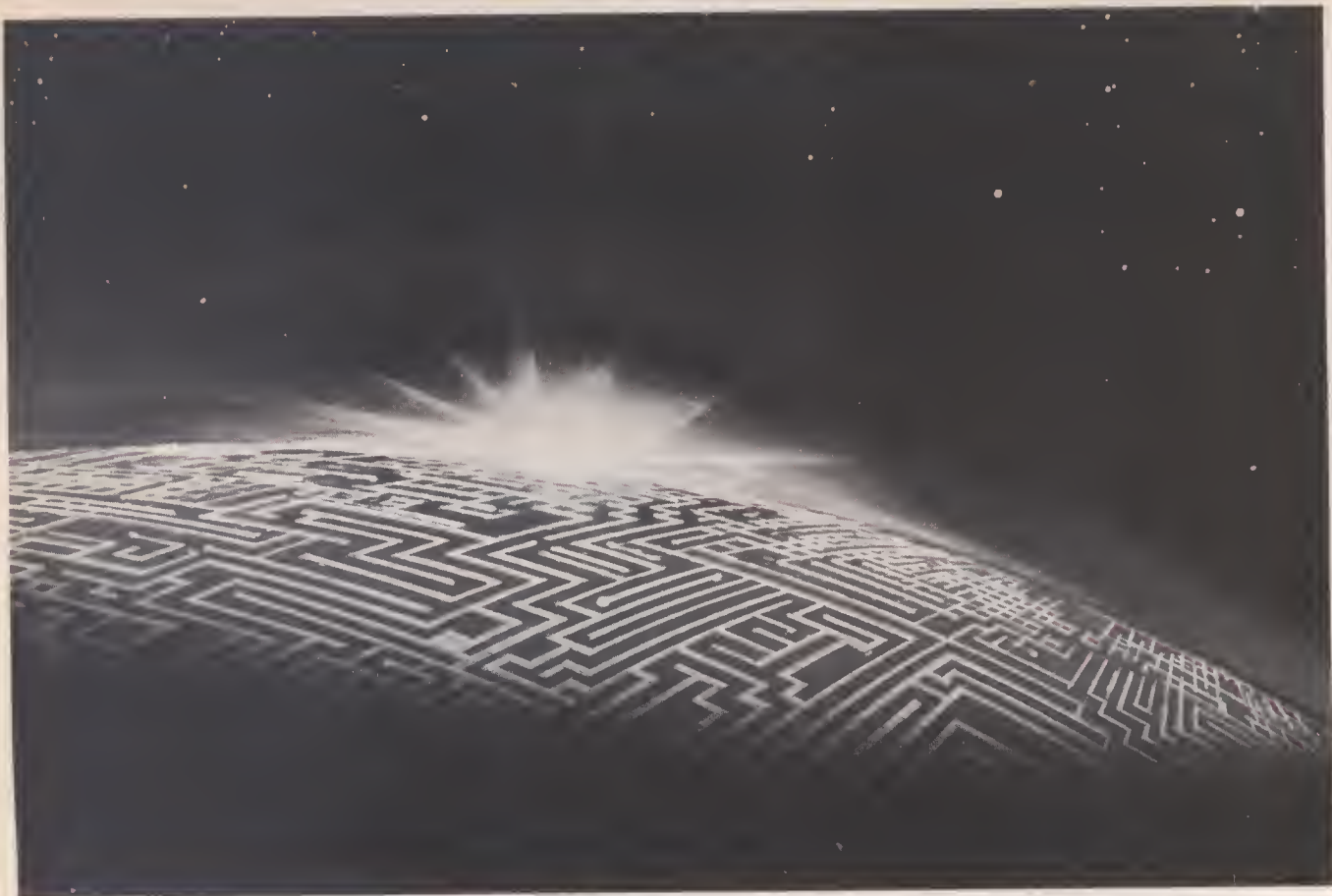
James Clayson

June \$19.95 paperback

Available at bookstores or directly
from The MIT Press. Call (617) 253-2884
MasterCard and Visa accepted.

The MIT Press

55 Hayward Street Cambridge, MA 02142



BE PRESENT AT THE DAWN OF CREATION.

The computer scientists and engineers at Lockheed Missiles and Space Company are now entering a bold new era in computer technology. An era full of opportunity, as we explore the full potential of artificial intelligence. To achieve that goal, we are establishing a new Artificial Intelligence Research Center in Palo Alto, California.

This Center will be dedicated to pure research into areas such as image and speech understanding, natural language understanding, knowledge-based and model-based systems. The Center will also support training, consulting and hardware/software tool evaluation programs for the entire company.

We're building a team of skilled professionals to meet the challenges ahead. This will truly be a ground floor opportunity. So if you're interested in defining tomorrow's technologies and are degreed in computer science, or an AI related field, we invite you to explore this unique opportunity.

 **Lockheed Missiles & Space Company**

Innovation
Giving shape to Imagination.

RESEARCH

- Evidential Reasoning
- Reasoning with Uncertainty
- Reasoning in Context
- Machine Learning
- Knowledge Acquisition and Representation
- Data Fusion
- Human-Machine Interface

APPLICATIONS

- Maintenance and Diagnostics
- Satellite Autonomy
- Situation Assessment
- Data Interpretation

For immediate consideration forward your resume to Professional Staffing, Dept. 530GAJS, Lockheed Missiles & Space Company, P.O. Box 3504, Sunnyvale, CA 94088-3504. We are an equal opportunity, affirmative action employer. U.S. citizenship is required.

The Architecture of FAIM-1

Judy M. Anderson, William S. Coates, Alan L. Davis, Robert W. Hon,

Ian N. Robinson, Shane V. Robison, and Kenneth S. Stevens

Schlumberger Palo Alto Research

FAIM-1 can be scaled to provide two to three orders of magnitude improvement over conventional AI machines with minimal inconvenience to programmers.

This article describes a symbolic multiprocessing system called FAIM-1.* FAIM-1 is a highly concurrent, general-purpose, symbolic accelerator for parallel AI symbolic computation. The paramount goal of the FAIM project is to produce an architecture that can be scaled to a configuration capable of performance improvements of two to three orders of magnitude over conventional architectures. In the design of FAIM-1, prime consideration was given to programmability, performance, extensibility, fault tolerance, and the cost-effective use of technology.

Programmability. Although the FAIM-1 machine architecture is unconventional, the software environment provides a concurrent programming language and an application development system that are based on models familiar to members of the AI community. This environment permits immediate evaluation of the FAIM-1 architecture when that architecture is used for existing applications, and it eases the burden on programmers of future applications.

Performance. Effective use of concurrency is the primary mechanism employed by the FAIM-1 system to increase significantly performance over conventional sequential systems. Hardware concurrency is exploited in the operation of

- the individual processing elements,
- subsystems within the processing elements, and

*Work on the FAIM (Fairchild AI Machine) project was begun at the Fairchild Research Laboratories and has since moved to another research lab within Schlumberger.

- components within each subsystem.

Software concurrency is exploited in two distinct forms. *Spatial concurrency* involves a set of independent tasks, each working on a partitioned piece of the problem. *Temporal concurrency* involves pipelined execution in which stages can be viewed as concurrent tasks operating on elements of a stream of data at different times.

Extensibility. High priority was given to creating a design permitting arbitrary expansion of the hardware resources. Expansion in FAIM-1 requires minimal rewiring, and no modification to either the user software or system software. The communication topology is planar, and therefore will not become a liability as technology advances to permit evolution from multiple-chip to single-chip processing elements. Wiring complexity scales linearly with processing-element count. All hardware module interfaces are self-timed¹ to permit individual components to evolve independently in performance, implementation technology, and functionality. Self-timed circuit design is a type of circuit design discipline that does not use a global clock to guarantee synchronization. In this style, each component keeps time internally and provides interface handshaking signals to coordinate with its partner subsystems. The result is an architecture that is easy to modify and exhibits a favorable cost/performance ratio under scaling.

Fault tolerance. Any solution to the fault-tolerance problem inherently contains redundancy. The FAIM-1 contains significant redundancy, both in terms of

processing elements and in the way these elements are interconnected. FAIM-1 is designed to be fault tolerant at the processing-element level, but not at the gate or circuit level. The resource-allocation mechanism permits the reassignment of tasks to processors, the message-routing algorithm is capable of routing messages around failed paths and processors, and the system software supports self-diagnosis.

Technology. The architecture is designed to take advantage of the cost and performance of both advanced VLSI circuit technology and advanced packaging technology (immersion-cooled wafer hybridization is an example of the latter) as they become available.

The focus of this article is on the physical architecture of the FAIM-1 system. However, to understand some of the design decisions, one must examine the salient aspects of both the software system structure and the programming language that the physical architecture supports. The next section ("Software structure") presents a synopsis of both topics; subsequent sections present the hardware architecture.

Software structure

The architecture of the FAIM-1 serves as a high-speed evaluation engine for the concurrent programming language OIL (Our Intermediate Language), and supports a style of distributed multiprocessing system structure that is embodied in the runtime operating system.

The OIL language and OIL objects. OIL is a high-level, concurrent, symbolic-programming language. The design of OIL was influenced by current AI programming practices, in which a number of AI programming languages are widely used. A distillation of these languages leaves three main linguistic styles: object-oriented programming, logic programming (primarily Prolog), and procedural programming (primarily Lisp). A complex AI application may require several (or all) of these programming styles. Emulating one programming style within another is inefficient, so there is a need for a better linguistic mechanism, one that efficiently incorporates or efficiently supports the essential features of the major styles. OIL has been designed to provide concurrent versions of each of the three linguistic styles.

An OIL program is a collection of objects that communicate by sending messages. The nature of the communication structure explicitly indicates the top level of concurrency represented by the program. Individual objects may themselves be concurrent program fragments. An OIL object consists of some local state information (typically in the form of variables) and several *ports* through which messages are sent and received in FIFO order. A *behavior*, associated with each port, describes what the object does in response to a message. The behavior is a program that may modify the local state and/or send messages to other objects. Atomic OIL objects are of two distinct types: *logical* and *procedural*. Logical behaviors are written in a declarative style similar to a parallel version of Prolog.² Procedural behaviors are written imperatively in a lexically scoped dialect of Lisp that is similar to T.³ Objects can be nested heterogeneously to form other objects that permit control to pass between declarative and imperative behaviors.

An OIL object consists of

- *state*, which is represented as the collection of variables and data structures that are considered local to the object;
- *ports*, that is, the set of entry points that may receive messages;
- *entries* (an entry is a subset of ports); and
- *behaviors*, that is, code that may be either procedural or logical.

Objects can be created dynamically or statically. Upon creation, the state of an object is set to its initialization value. An object takes action when a message arrives. To support distributed procedure calls and parallel process synchronization, subsets of ports may be grouped into *entries*. Behaviors are associated with entries rather than ports. When all of an entry's ports have a message, then the associated behavior is said to be *fireable*. In cases where only a single port's message invokes a behavior, that port is also labelled as an entry. In cases where an object is defined in terms of other objects, the inheritance is static.

An object may consist of an arbitrary number of behaviors and ports, and each behavior is viewed as a potentially independent code fragment. Since the object's state is accessible by any of that object's behaviors, a potential source of nondeterminacy exists. To prevent this situation, the behaviors are viewed as a set of mutually exclusive transactions. When a behav-

ior is started up, all other behaviors are inhibited until that behavior terminates. Furthermore, a behavior may close ports and thereby temporarily inhibit message delivery on those ports until they are subsequently re-opened.

The roles of logic and procedural components are quite distinct. The logic component is used to express in a succinct manner nondeterministic pattern-driven search, while the procedural component is used for sequential algorithm specification, manipulating unique objects, and expressing history-sensitive algorithms. Neither component projects its semantics on the other. Therefore, procedural objects do not have multiple versions of their environments, and logical objects do not have changeable state variables. Communication between logical and procedural objects is based on message streams managed by manipulation of continuations in the sending and receiving objects. The exact semantic significance of continuations is somewhat different on each side, but both use continuations as "handles" to obtain subsequent values in a stream. A logic component accepts an input stream of goals, and produces an output stream of solutions. In general, there are several logical solutions per goal. A procedural component accepts an input stream of function calls, and produces an output stream of response messages, usually with one response per input function call.

The binding of variables observes the semantics of the object in which the variables are defined. Thus, a logical variable has multiple alternative bindings, in keeping with the nondeterministic semantics of logic objects, and a procedural variable has values that are changed by direct assignment. Moreover, components are not allowed to violate the binding policies imposed by the variable's proprietor. Bound logic variables are invisible to procedural accessors; instead, such accesses directly return the variable's value. Unbound logic variables are detected as such in procedural components, but cannot be bound by them. Procedural variables are passed by value to logic components when used in parameters. Hence, procedural variables (as assignable entities) are invisible in logic components.

The programmer may also annotate OIL code with *pragmas*, which describe some of the expected runtime dynamic behavior of the code. These programmer-supplied hints are used by the static resource allocator to partition code and

data onto the physical resources of the machine. The pragma information gives the programmer control over some aspects of the allocation strategy. If no pragma information is supplied, the program will still run, although perhaps not as efficiently.

Procedural OIL. The procedural component of OIL is a parallel modification of a lexically scoped dialect of Lisp called T.³ The primary modifications to T facilitate the use of concurrency; they include a concurrent reformulation of the basic semantics, and the addition of parallel control and data structures, operations on parallel data structures, and specifiable evaluation strategies that permit a variety of parallel evaluation methods. The primitive special forms are exactly as defined by T with two exceptions. The first exception involves *Cond*, which is the normal T conditional. *Cond* also exists in OIL, but *Cond* = has been added, and is the parallel OIL version. *Cond* = causes the guards to evaluate in parallel, and the first one that evaluates to True is pursued. Semantically, this implies that an arbitrary evaluation of "True" by the guard forces selection, since the notion of what is first is not controllable by the programmer. The second is to permit special forms for specifying pragmas and type information. These are, respectively, the *Pragma* and *Proclaim* special forms. Pragmas fall into three categories:

- estimates of the probability of taking a particular branch in a decision,
- estimates of the size of dynamic data structures, and
- hints about appropriate allocation decisions.

In all other respects procedural OIL is isomorphic to T.

Logical OIL. Logic behaviors are written in a parallel form of Prolog that is syntactically similar to DEC-10 Prolog. The same notation is used for clauses, lists, and several "evaluable predicates." In general, operations such as arithmetic functions, which do not have an inherent sequential semantics, are the same in both languages. Operations like assert and retract are not supported.

An OIL logical program consists of a set of named objects, each of which contains any number of clauses. The names of the objects serve as names of *worlds*. The logic objects can be nested, giving the effect of additional worlds. The programmer may indicate a goal as *solve(x, W)*, meaning that goal *x* is to be solved in the context of

world *W*. This implies that the rules for solving *x* are defined in world *W*. An inner world inherits all the rules of the outer worlds. If the programmer omits the world specification, then a default single-world model is used. The programmer may also indicate private rules that are not inherited by inner worlds. Logical objects differ from procedural objects in that they cannot have internal state variables. Programmers must use procedural code to describe operations that modify an object's local state. Variable bindings invoked under unification can affect local state only after they are passed back to a procedural module, which assigns the value to one of the object's state variables.

Clauses of logic programs are compiled into sets of primitive processes. These primitive processes are objects that use local state variables to represent a portion of the global runtime environment. The processes respond to incoming messages by changing state and generating messages for other processes. The two types of primitive processes are *AND processes* and *OR processes*. *AND processes* execute the bodies of nonunit clauses, and *OR processes* manage execution of procedures that are defined by more than one clause. In the compiled logic program, there is at least one *AND process* for the right hand side of each nonunit clause, and at least one *OR process* for each *procedure* (set of clauses with heads that have the same functor and arity). An overall view of the computation would show an *AND/OR process tree*, with *AND processes* creating *OR descendants* to solve each literal in a goal statement, and *OR processes* creating *AND descendants* to solve bodies of matching clauses.

In *AND processes*, the basic actions, the internal states, and the reaction to a particular message depend on the desired control model. The programmer may provide mode declarations on logical variables; for example, the literal *p(x?, y?)* implies that when *p* is evaluated, it will consume a binding for *x* and produce one for *y*. The parallelism in logical OIL is primarily the nondeterministic pattern-directed search mechanism that is obtained through *OR parallelism*.⁴ Limited forms of *AND parallelism* similar to that proposed by DeGroot⁵ are also provided in cases where independent *AND processes* can be identified at compile time or by a simple runtime ground check. The use of mode declarations significantly enhances the programmer's ability to control the amount of *AND parallelism* that can be exploited.

Interfacing logical and procedural behaviors. The interface between the two types of behavior components is essentially a form of procedure call. A logical behavior calls a procedural behavior by means of an evaluable predicate that is syntactically identified. The primary difference in the semantics is that the solution of a logical subgoal by a procedural behavior may succeed more than once. In this sense procedural behaviors behave more like normal subgoal solutions than like conventional Prolog built-in predicates. Subsequent calls to the procedural component will result in *next* messages that will retrieve the next stream element.

A procedural behavior may call a logical behavior by sending it a *solve* message containing a goal and a world in which that goal is to be solved. Calling the continuation of the logical behavior will retrieve the next element in its solution stream.

Resource allocation. OIL programs are compiled into object code on a host Lisp Machine. The host then downloads the object code onto the FAIM-1 for execution. Critical decisions about where to load individual objects are made during a phase of compilation called *resource allocation*. The resource allocation process involves balancing the use of the parallel execution hardware with the cost of runtime communication overhead. The resource allocation phase permits the writing of programs, even if the writer lacks a detailed understanding of the hardware, interconnection structure, or communications costs.

In general, there are three basic approaches to resource allocation:

- *Programmer-defined allocation*, which can be implemented either as part of the programming language or by an alternative description, places responsibility for making all resource allocation decisions on the programmer.
- In *dynamic allocation*, the overall processor utilization must be measured at runtime by the system. By means of this analysis, the workload is adjusted during program execution.
- *Static allocation* involves analyzing the source program and partitioning it into a set of allocatable tasks in a way that maximizes processing concurrency while minimizing overhead from interprocessor communication.

The complexity of resource allocation for large programs makes it unlikely that

programmer-defined allocation will be a viable long-term solution. Dynamic allocation inherently implies significant levels of runtime overhead. Hence, the primary focus for FAIM-1 is on static methods. The OIL programmer can influence the static allocator by special annotations (see the discussion of pragmas, above), and some simple dynamic load balancing can be performed when runtime conditions indicate that it is necessary.

Hardware structure

The FAIM-1 architecture consists of a number of independent processing elements, called *Hectogons*, interconnected in a hexagonal mesh. The following sections describe the overall structure of the machine and provide insight into some of the design decisions made in creating it.

Communication topology. Many possible multiprocessor interconnection schemes are currently being investigated. See, for example, the current research efforts on the Cosmic Cube,⁶ Butterfly,⁷ and DADO.⁸ Desirable topology characteristics include high performance, reduced wiring complexity, flexibility, fault tolerance, and simplicity of implementation. Furthermore, a goal is that these properties remain attractive under scaling.

The topology used in the FAIM-1 is a hexagonal mesh. Processing elements communicate directly with six neighbors; the processing elements themselves are organized into hexagonal *surfaces* that are combined in a similar six-neighbor fashion.

When wires leave a processing surface through the processing elements at the periphery, they are folded back onto the surface in a three-axis variant of a twisted torus. In Figure 1, the basic topology is illustrated, along with the wrap lines and switches that complete the interconnect structure. For purposes of illustration, only a single, wrapped axis is shown; in the complete topology, all edge ports are connected, requiring two additional sets of wraps like the one shown in the diagram.

This particular wrapping scheme results in a simple routing algorithm and provides a minimal switching diameter for a hexagonal mesh. All PEs are viewed as if they were the center PE of the surface, and routing decisions are based on three-axis relative coordinates. This simple algorithm is implemented in custom hardware for performance reasons.

Each peripheral port communicates with an off-surface device, as well as being wrapped back to the opposite edge of the surface. These off-surface connections permit communications with I/O devices and with other surfaces. The external connections are made by introducing a simple three-way switch, which is shown in Figure 1. Communication with other processors on the surface is via the three-way switch, which routes signals back to the other edge of the surface. The switches have three ports:

- internal (for local surface messages),
- external (for adjacent surfaces, I/O devices, or the host), and
- wrap (for the local surface via the wrap line).

Switching decisions are based on which of the three ports a message arrives on and the destination contained in the message header.

The size of a surface is defined by the number of processors n on each edge of the hexagonal surface. The surface is referred to as an E - n surface; the number of processors in a surface scales as $3n(n-1) + 1$. For example, an E -3 processor surface has three processors on each of the six edges and contains a total of 19 processors. For surface sizes between E -1 and E -7 inclusive, the number of PEs is a prime number, which is advantageous from the standpoint of fault tolerance and initialization.

A hierarchical instance of a FAIM-1 processor is built by tessellating multiple hexagonal surfaces. Locality among groups of processors is increased and the communication diameter of the system is decreased as compared with a single-surface instance that has the same number of processors. These properties can be demonstrated by the following example. An E -7 surface contains 127 processors and has a diameter of six, while seven E -3 surfaces can be tiled to form an S -2 E -3 machine (shown in Figure 2) that contains 133 processors with a diameter of five. The switching diameter improves dramatically as the processor count is increased. A 58,381-processor E -140 has a diameter of 139, while a 58,807-processor S -9 E -10 FAIM instance results in a diameter of 89, a full 50 hops better (worst case) than the single E -140 surface.

The probability of component failure in a system statistically increases as more components are added to the system,⁹ making fault tolerance an important aspect of highly replicated architectures. Fortunately, distributed ensemble architectures intrinsically contain redundant

elements that can be used to support fault-tolerant behavior. Koren¹⁰ has shown that hexagonal meshes are particularly attractive fault-tolerant topologies. In addition, fault-tolerant message routing is possible because of the multiplicity of paths over which a message may be routed to its destination.

In the FAIM-1 machine, all of the topology-dependent hardware is contained in a single subsystem called the *Post Office*, which is described in detail in a later section. The remainder of the machine is independent of connection topology, and could be utilized in other connection schemes.

The Hectogon. The processors located at each node in the communication topology are called *Hectogons*. A Hectogon can be viewed as the homogeneously replicated processing element of the FAIM-1 architecture on one hand, and as a medium-grain, highly concurrent, heterogeneous, shared-memory multiprocessor on the other. Internally, each Hectogon is constructed of several subsystems or coprocessors, all of which may be active concurrently.

This double view is the result of the consistent exploitation of concurrency at all levels of the FAIM-1 system, and is motivated by our belief that the scalability of a multiprocessor architecture is of prime importance. Performance of an architecture as it is scaled up is critically affected by four factors:

- the performance of each individual processor,
- the average percentage of processors that are active,
- the efficiency of interprocessor communication, and
- the total number of processors in the aggregate machine.

The intent of the FAIM-1 project is to pursue aggressively each of the four aspects by designing a powerful processing element that permits high levels of replication.

Individual coprocessors directly support logic programming, parallel Lisp, and complex runtime system duties, such as task switching and scheduling. Within each coprocessor, other linguistic features are supported by specific aspects of the architecture. For example, the evaluation processor contains parallel tag hardware to support the polymorphic function-calling nature of Lisp. Rather than allocating a single task to each processing element and risking a low percentage of

active processors, FAIM-1 allocates a number of parallel tasks to each processing element. Tasks can be complex program fragments requiring a wide range of computational support. This strategy improves processing element utility by increasing the probability that a runnable task will exist at any given time. Inter-process communication is facilitated by including a high-performance message-handling coprocessor (the Post Office). A large number of Hectogons can be tiled together (as described in the section on "Communication topology," above) to form a fully distributed (that is, with no shared memory or control) multiprocessor system.

The Hectogon's subsystems are connected by an asynchronous System Bus (SBus), and by custom interfaces in some cases. Subsystems communicate with each other by means of a flexible, speed-independent signalling protocol. The self-timed behavior of the subsystems allows them to be independently tuned in terms of both performance and function without impacting the designs of the other subsystems. While each of the six subsystems is a reasonably general system-level component for distributed ensemble architectures, the particular instantiation of each has been tailored with a specific view of the Hectogon in mind. A Hectogon's subsystems and their interconnection are shown in Figure 3.

The six subsystems connected by the SBus are

- *Evaluation processor (EP)*. A streamlined, non-microcoded, stack-based processor responsible for evaluating machine instructions.
- *Switching processor (SP)*. A small context-switching processor that is responsible for interpreting the run-list in data memory (the Scratch RAM) and moving blocked contexts out of processor registers and into SRAM process-control blocks, and then loading a new, runnable context into the processor registers.
- *Instruction stream memory (ISM)*. A specialized instruction memory that not only stores the instructions, but is also responsible for finding the appropriate instruction, partially

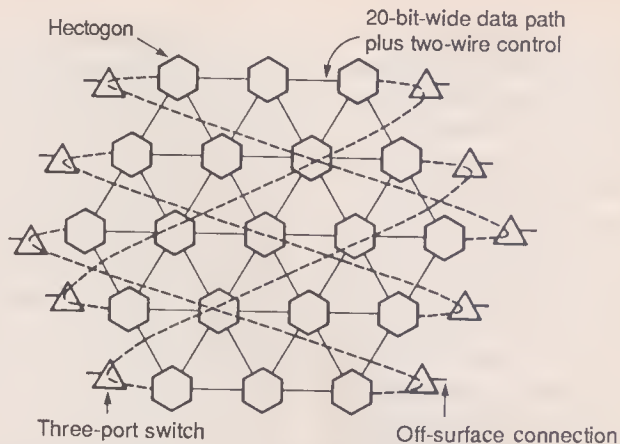


Figure 1. E-3 surface with three-way switches. (Reprinted from the *Proceedings of the Sixth International Conference on Distributed Computing Systems* © IEEE.)

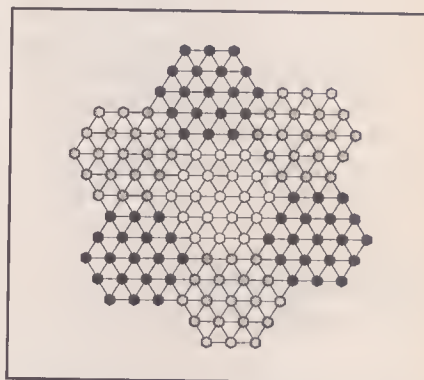


Figure 2. S-2 tessellation of E-3 surfaces. (Source: *Proceedings of the 1985 International Joint Conference on Artificial Intelligence*, 1985. Used by the courtesy of Morgan Kaufman Publishers.)

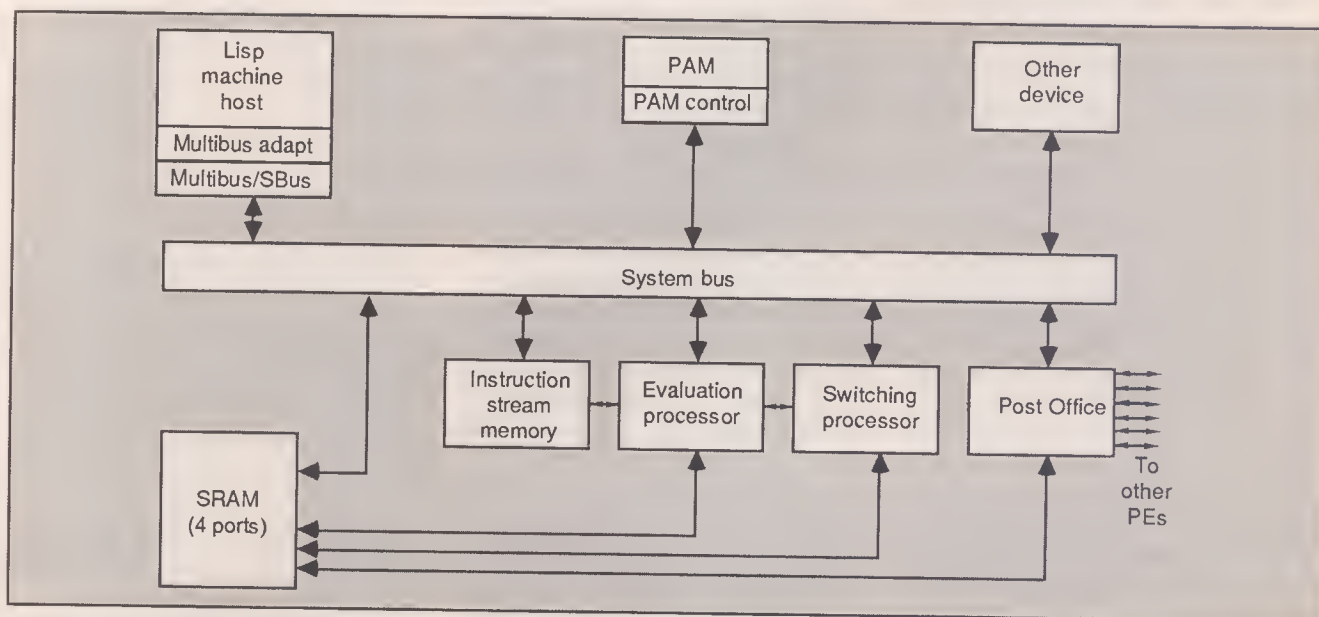


Figure 3. Block diagram of a Hectogon.

- decoding it, and handing it to the EP.
- *Scratch random access memory (SRAM)*. The SRAM is the local-data memory of the Hectogon. It is a four-ported subsystem that provides concurrent access to the EP, SP, SBus, and Post Office.
- *Pattern-addressable memory (PAM)*. A parallel associative memory system capable of matching complex structures, such as S-expressions, in which *don't cares* may exist either in the query or in the data. In the present instantiation, the match function is "mock unification," that is, the extent to which unification can be done without dereferencing bound logical variable pointers.
- *Post Office*. An autonomous communications subsystem responsible for all aspects of the physical delivery of inter-Hectogon messages. The Post Office is the only topology-dependent coprocessor in a Hectogon.

The remainder of this article presents each coprocessor in some detail.

A two-processor partnership

The processor is designed to be a high-speed evaluator of OIL programs. As such, it contains specific features that provide efficient support mechanisms for object-oriented, procedural, and logic programming. Processor utilization on the FAIM-1 is improved if a small number of independent, and therefore concurrent, tasks are allocated onto a single Hectogon. This increases the probability that for any particular Hectogon, a runnable task will be available at any given time. This imposes a need for a processor to support multitasking and rapid context switches between tasks.

To achieve the goal of rapid context switching, the processor is implemented as a partnership of two processors; the two processors are called the *evaluation processor (EP)* and the *switching processor (SP)*. The EP evaluates the currently active task, while the SP sets up the next runnable task. Tasks run in a context-specific register set; the processor contains two context-register sets. Each set contains the stack buffers, processor status word, general-purpose registers, and so on. One set is marked *active* and is used by the EP, while the other is marked *idle* and is used by the SP to save the old context and load the next context from process-control

blocks stored in the SRAM. This concurrent and cooperative partnership is also supported by the SRAM design in that both the EP and the SP have separate dedicated ports to the SRAM. It typically takes a single instruction time to switch contexts. This assumes that the SP has completed its next-task prefetch activity. If this is not the case, then the EP will halt* until the new context becomes valid.

Normally there are four ways that the currently running task can be suspended: an interrupt, a trap, task termination, or task block. A task block can occur at the program's request, or when an unforeseen delay is incurred, for example, when a message sent to a remote processor requires a reply before the sending task can continue.

In FAIM-1, the same context sets and switching mechanism are used for all four cases. There is no special supervisor or kernel context. In both task termination and task blocking, the next reasonable action is to run another task, as nothing more can be done on the evaluation of the current task. Since traps usually require information from the current context (for example, the arguments to the trapped instruction require modification by the service routine), trap service routines are run in the current context in a manner similar to that of a simple function call. Interrupt-driven tasks inherently start and terminate cleanly, and therefore can also be evaluated in the current context.

The EP. The EP executes instructions that it receives from the ISM by making use of the active-context register set. To reduce the complexity of the EP and achieve high performance, a stack-based RISC¹¹ organization was chosen. The need for a long word length does not exist, since the address space of the local memory is small and since the focus of the architecture is on symbolic processing rather than large-value numeric computation. The EP is further simplified by the existence of the ISM, which selects, formats, and delivers the proper instruction to the EP.

The existence of a separate instruction delivery subsystem (the ISM) permits a simple two-stage pipeline to be employed for the EP. The first stage supports instruction decode, operand selection, and modification. The second stage performs

the ALU operation, modifies the result if necessary, and stores the result. The ISM runs concurrently with the EP and effectively contains another two pipeline stages that perform effective-address translation, instruction fetch, and partial instruction decode.

The EP datapath is 28 bits wide. Eight bits are reserved for tags that are processed in parallel by separate tag-manipulation hardware. The tags are analyzed concurrently with the processing of the 20-bit data field, and may generate a trap when the operand tags are inappropriate for a particular operation.

The EP also contains a full 20-bit wide ALU that supports logical operators, integer arithmetic, and shifting of instructions.

The EP instructions combine the small instruction size of stack-organized processors and the streamlined style of the RISC methodology. This is achieved by trading register-based arithmetic and logical operations for stack-based operations. In addition, the instruction set supports a simple load-and-store model, which may also be indexed from any of the general-purpose index registers.

The instruction set is tuned so that most instructions run in a single cycle. The only instructions that consume additional cycles explicitly are instructions that generate memory references, and interface instructions to the other subsystems. However, any instruction may trap and thus effectively consume additional cycles. In this case, the instruction is completed whenever the target trap routine returns.

The result is a simple RISC-processor that provides significant support for the efficient evaluation of OIL programs and that is physically small enough to permit the high levels of replication required for a cost-effective, high-performance FAIM-1 PE.

The SP. The context switcher is a small processor that loads values found in a process control block stored in the SRAM into the idle context-register set and unloads them, too; it thus permits context switching without stealing cycles from the EP. The main data structure that the SP manipulates is the *run-list*. The run-list is produced by the scheduler, which is an EP task that is run when necessary. The run-list is a linked list of process-control blocks (PCBs)** in which each PCB contains a pointer to the next PCB in the list.

*Actually, the EP is not explicitly aware that the SP is not ready. The self-timed interface between the EP and SP will automatically and transparently inhibit the EP from evaluating an invalid context.

**The actual implementation is more complex than this, but additional detail is omitted here for clarity.

In blocking, a process calls a system routine that requests that the SP perform a context swap and save the process's state. (Depending on why the process is blocking, it may be moved to the blocked-list or may remain on the run-list.) The SP immediately switches to the other set of context registers, and the EP can execute instructions from the new context. The SP then moves the contents of the now idle context register set to the appropriate task PCB in SRAM, examines the run-list to find the next runnable task, moves its context from the PCB to the idle context register set, and sets the proper context validity flag. The SP will halt at this point and remain halted until another context switch is signalled.

Arriving message traffic from other Hectogons may cause idle processes to be rescheduled as runnable tasks. The message handler and scheduler support this by linking arriving messages with their target tasks and moving them to the run-list. The scheduler can therefore be invoked by a "message-delivered" interrupt from the Post Office or host machine.

The instruction stream memory

The instruction stream memory (ISM) is one of the specialized "smart memories" that are employed in the FAIM-I processing element, and is designed to deliver instructions to the processor at high speed. To accomplish this task efficiently, the ISM performs several functions that are usually performed by the processor in a conventional system. These include calculation of instruction addresses, processing of branches and subroutine calls, and handling of traps and interrupts.

Implementing a separate instruction delivery module can have several advantages, as is demonstrated in the Dorado architecture.¹² Since only instructions are stored in the ISM, it can be optimized for its sole function of instruction delivery. Conventional solutions place a specialized intermediate piece of hardware between the processor and main memory, such as an instruction cache or translation look-aside buffer. This implies more complicated control and requires that relatively slow off-chip signals be driven twice. The ISM organization capitalizes on the co-placement of both the instruction memory and the instruction-delivery-control logic on the same VLSI device. Since on-chip bandwidth is usually an order of magni-

tude faster than off-chip bandwidth, a significant level of manipulation can be performed concurrently with processing activity in the EP. In addition, very wide bus widths are practical on-chip but impractical at chip boundaries, thus increasing the effective on-chip bandwidth.

In addition to the advantages of more efficient access to instructions, the design of the EP is simplified. The instruction-fetch stage of the pipeline has effectively been transferred to the ISM, along with the program counter. The resulting shorter pipeline in the EP increases the throughput of the processor and simplifies exception-handling duties.

The ISM capitalizes on the fact that instructions are not randomly accessed. In most programs, code is naturally partitioned into small sequences of linearly ordered instructions that terminate in branches or subroutine calls. These sequences are formalized in the FAIM-I machine and are called *tracks*. As one intuitively, instruction tracks vary in length; however, the ISM's instruction storage maps them onto a set of fixed-length physical tracks. Linkage information corresponding to control instructions is associated with the physical tracks in a header field.

Control point information is maintained in a *current track-address register (CTR)*, which is similar in function to the program counter of a conventional system. An individual instruction address consists of a track number and a track offset.

Branch processing. Execution of branch-type instructions is quite complex in most large machines, owing to the large number of instructions that can cause a branch and the wide variety of addressing modes allowed. In keeping with the streamlined RISC theme, the FAIM-I uses only one jump format, which is flexible enough to support conditional and unconditional branches, calls, and returns. Branch-addressing modes that require multiple accesses to memory (for example, indirect addressing) or branches requiring additional arithmetic operations (for example, indexed or decrement-and-skip-if-zero) are not implemented. Instruction decoding becomes simple both for the EP and ISM. The ISM examines each instruction as the instruction is prepared for delivery to the EP. When a jump is detected, the ISM autonomously and concurrently processes that instruction, continuing delivery with the first instruction

from the target stream.

Conditional branches are a canonical problem, and delay the delivery of source code in pipelined program execution. They are usually the main bottleneck in lookahead and prefetch strategies. In conventional systems, the problem is that by the time the branch is executed and the correct path is resolved, some succeeding instructions have already entered the pipeline. If the branch is taken, the current contents of the pipeline must be discarded, incurring a delay while the pipe is filled again. The standard method of keeping the pipeline full is to use a delayed branch instruction, as is done in the MIPS architecture.¹¹

In FAIM-I, the ISM decodes jump instructions before they even enter the pipeline, so a pipeline flush is not necessary and the delayed branch strategy is not applicable. This is because the required continuation of the instruction stream is always correctly delivered. There is, however, an analogous dependence problem in that the outcome of a conditional jump may depend on the result of a previous instruction that has not yet finished executing. To ensure that the branch condition has become valid by the time the ISM detects the jump, an *advance condition code set* method is used.

Subroutine calls and returns. A subroutine, or function call, in the OIL language compiles to a simple jump instruction and is processed by the ISM in almost exactly the same manner as a normal jump. The jump instruction format contains a "save" bit that is set for a "calling" jump and indicates that the current contents of the "program counter" (the CTR) should be saved in a special register called the *jump track-address register (JTR)* before the jump is executed. A return from subroutine consists of a Jump instruction specifying the current contents of the JTR as the target address. If the called subroutine needs to call other subroutines, then it must explicitly save and restore the JTR by making use of the control stack.

Exception handling. Traps and interrupts are additional factors that cause a break in the instruction stream. As such, they have an impact on the operation of the ISM. Traps correspond to error conditions arising during the course of executing an instruction, while interrupts are generated by some external device and are completely asynchronous with the program being run. In both cases, the current code

stream must be broken to permit the execution of the trap or interrupt-service routine. This is accomplished in a manner similar to that of a subroutine call.

ISM utility. In a conventional processing system, there is a large amount of undesirable traffic between the processor and main memory. Not only useful data, but information on where to find the data, and even information needed to compute this information, must be sent to and from memory.

In FAIM-1, one major class of memory references is associated with the access of the machine code instruction stream. Studies indicate that instruction fetches can account for 50 percent of all memory references on computers such as a VAX or IBM S/370. The single-chip organization of the ISM greatly reduces the severity of this bottleneck. By reflecting the structure of instruction tracks in the structure of the memory and providing integrated logic to perform in an autonomous manner common functions, such as branching, prefetching and subroutine calls, the design of the processor is simplified and new levels of performance are made possible.

The proposed design is simple enough to allow single-chip implementation; multiple ISM chips can be cascaded to provide a maximum-size instruction bank of one million instructions. The EP organization permits an arbitrary number of banks to be used in a single PE; however, the first prototype will contain only a single maximum-size bank per Hectogon.

The Scratch RAM

The SRAM is a four-ported, random-access memory that is the local-data memory system of the Hectogon. An SRAM contains 1 Meg of 28-bit data words. Each data word is further divided into eight tag bits and 20 data bits. The SRAM allows access to memory through any of its four ports, which may all be active concurrently. The ports are connected to the SBus, the evaluation processor, the switching processor, and the Post Office.

The Post Office

Operational responsibilities of the Post Office. The Post Office¹³ is an autonomous coprocessor of messages that is capable of delivering messages concurrently with program execution in the pro-

cessor. The FAIM-1 Post Office contains seven data ports: six to Post Office communication processors in topologically adjacent Hectogons, plus an internal port to the SRAM of the local Hectogon. There is also a system bus link to control communication with the EP and other devices in the Hectogon. All message-delivery control is handled autonomously by the Post Office, freeing the local Hectogon to process tasks rather than stealing EP cycles to support inter-Hectogon message traffic.

The Post Office is responsible for the physical delivery of messages across the communication topology. Communication is initialized by the EP, which generates a message header for and pointer to the variable-length message body and places them in the SRAM. The Post Office extracts the message from the SRAM and delivers it by means of *virtual cut-through*¹⁴ over the communication network. The receiving Post Office places the message in the destination Hectogon's SRAM and notifies its EP that a new message has arrived. Since messages may vary in length, the Post Office may break larger messages up into a series of fixed-length packets that are physically transmitted across the topology. These packets are delivered to their intended destination individually.

The normal mode of delivery is to route a message to its destination by means of a hardware routing algorithm. In richly connected topologies, such as the hexagonal FAIM-1 topology, it is possible for a packet to take multiple paths to a nonlocal destination. The routing algorithm calculates the shortest paths and secondary paths to the destination. General communication efficiency can be significantly enhanced with a routing mechanism capable of detecting congestion and of routing packets around such congested areas in the communications network. All routes for packets in the Post Office are chosen dynamically at delivery time by prioritizing the paths and selecting a path according to the state of the buffers and the network. This mechanism can also be extended to introduce fault tolerance, since messages can be routed around non-functional nodes and ports.

The capability of dynamically routing packets around congestion and failed components implies that the order of packet arrival may vary from the order in which the packets were sent. The capability of reordering packets at their final destination is essential to ensure both deterministic behavior and to permit prop-

er reassembly of multiple packet messages. Although this increases the amount of work required at the receiving Post Office, it reduces congestion and failure-related message delays.

From the EP's point of view, messages are assumed to be delivered error-free. This assumption places the burden on the Post Office to verify checksums for each packet, organize retransmission when an error is detected, and synchronize packet-copy destruction in a way that ensures that at least one correct copy of the packet exists in some Hectogon. Positive and negative acknowledgment messages can be automatically echoed back to the sender from the destination Post Office upon receipt of a message. To avoid deadlock, the Post Office prevents unrestricted incremental resource claiming, thereby ensuring that each Post Office will not be congested indefinitely.

Post Office components. The Post Office is constructed from three types of basic components—*port controllers*, *buffer controllers*, and a *buffer pool*—as shown in Figure 4.

Most of the functions of the Post Office are performed by independent port controllers to enhance concurrency. The *packet ports* transmit data to an adjacent Hectogon across the topology, while the *PE port* stores and retrieves data from the local SRAM. The buffer pool acts as a temporary storage site and queue for messages that have arrived at the destination or require intermediate buffering between the source and destination nodes.

All seven ports share the same internal buffer pool. The function of the buffer controllers is to arbitrate and control access to this limited shared resource. The controllers are responsible for freeing space in the buffers by matching buffered packets with free ports on the path to the packets' respective destinations. The dynamic behavior of packet delivery is further enhanced by the buffer controllers, which access the buffers in a fair but unordered fashion.

Performance. Each of the seven ports of a Post Office node can potentially be transmitting data simultaneously. A pessimistic estimate of 20 MHz cycle time per port yields a burst communication bandwidth of 2¼ gigabits per second per processor-Post Office pair. An E-3 FAIM instance with 19 Hectogons will have a total peak communication bandwidth of approximately 25 gigabits per second. The

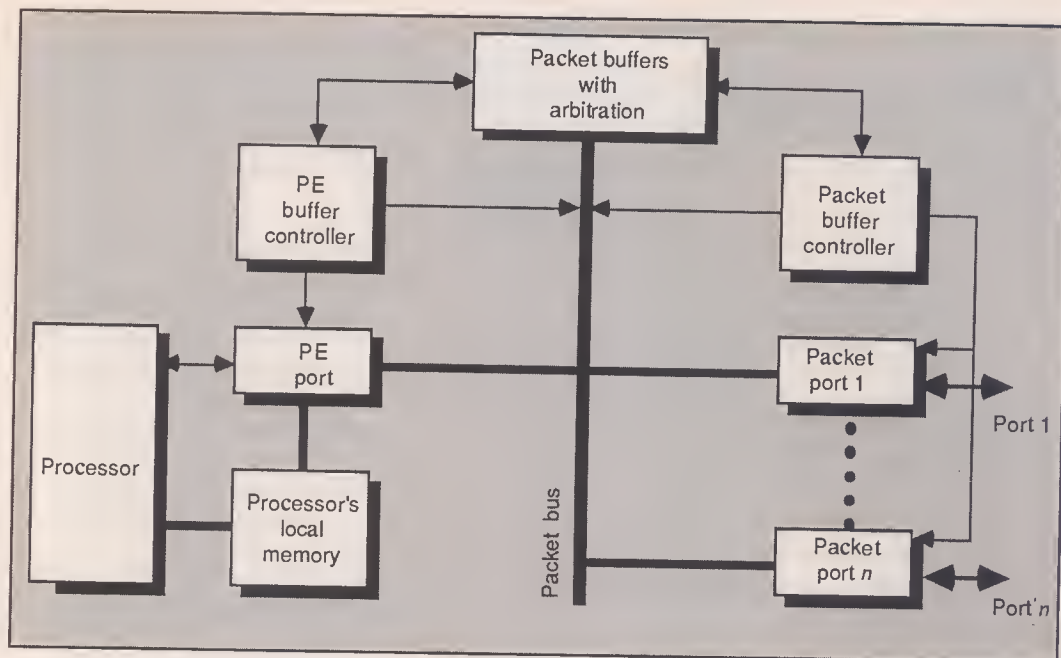


Figure 4. Block diagram of the Post Office.

speed-independent nature of the communication links automatically avoids synchronization and timing problems when peak demands are greater than buffering capacity or are owing to resource conflict. Such a high bandwidth is desirable for experimenting with machine scalability and supporting large processor counts.

In general, it is our belief that direct support for communications will be required in MIMD multiprocessors with many processing sites. This requires a significant reallocation of the transistor budget to achieve the proper balance between processing and communication performance. The Post Office is an experiment that couples a scalable topology and architecture with a communication coprocessor that is completely responsible for the physical delivery of messages. By operating concurrently with program execution and allowing all ports to transfer data simultaneously, the performance of the Post Office should be sufficient even for large-scale networks of processors.

The PAM

Pattern matching and logic programming. One of the most common functions used in AI programming is pattern matching (that is, matching on the LHS of rules, database searches, and so on). We feel that special pattern-matching support is necessary to achieve high-performance symbolic processing. As elsewhere in the Hecto-

gon, this support takes the form of a specialized memory component, here called the *pattern-addressable memory* (or *PAM*). The PAM is designed to store and match on S-expression structures of symbols and words, in much the same way that a traditional content-addressable memory (CAM) deals with single words. Like a CAM, the PAM does its matching in parallel over the whole of its stored contents.

The logic-programming part of OIL is based on Prolog, in which the execution mechanism, unification, is a special form of pattern matching. The PAM can therefore be used as a high-performance subsystem to support the evaluation of logical OIL behaviors. In a wider sense, the declarative semantics of Prolog provides a way of indicating data to be pattern matched and a means to control the application of that pattern matching.

PAM operation. The PAM consists of a number of PAM chips and the PAM controller, the latter providing the interface with the rest of the Hectogon. The PAM is used to store the heads of logic-program clauses as a linear array of symbols. Each clause head ends in a pointer to the compiled code area for that clause. When a logic program is evaluated by the EP, it sets up the goal to be solved in the SRAM; makes a call to the PAM controller; and then blocks, awaiting the results. The PAM controller, accessing SRAM over the SBus, fetches the goal and enters it into the PAM chips. The goal is entered symbol by

symbol and the match computed for all the clause heads stored "on the fly." The PAM contains circuitry to detect the absence of any matches as soon as that condition occurs. In that case, matching stops and control of the process returns to the EP.

Otherwise, the controller puts a list of the code pointers from the matching expressions in the SRAM, and reports to the EP that it has completed the task. The EP then uses the code indicated to complete the unification with the clause head. If successful, it then processes the clause body.

The most obvious function of the PAM is, therefore, as a clause-indexing device. Pattern matching is, however, more than just a useful means of clause selection; it is also a significant part of the unification algorithm (*unification = pattern matching + the logical variable*¹⁵). The matching that takes place in the PAM amounts to full unification for all those terms not involving variables. Recording and checking variable bindings is, however, outside of the PAM's scope. Such "bookkeeping" is therefore left to the EP, which acts as a unification post-processor.

The PAM memory chip. Each PAM chip consists of a number of blocks, each of which in turn consists of storage and logic parts. The storage part contains the symbol name and special tag and status bits used in the matching operation. In the current implementation, 10 such words

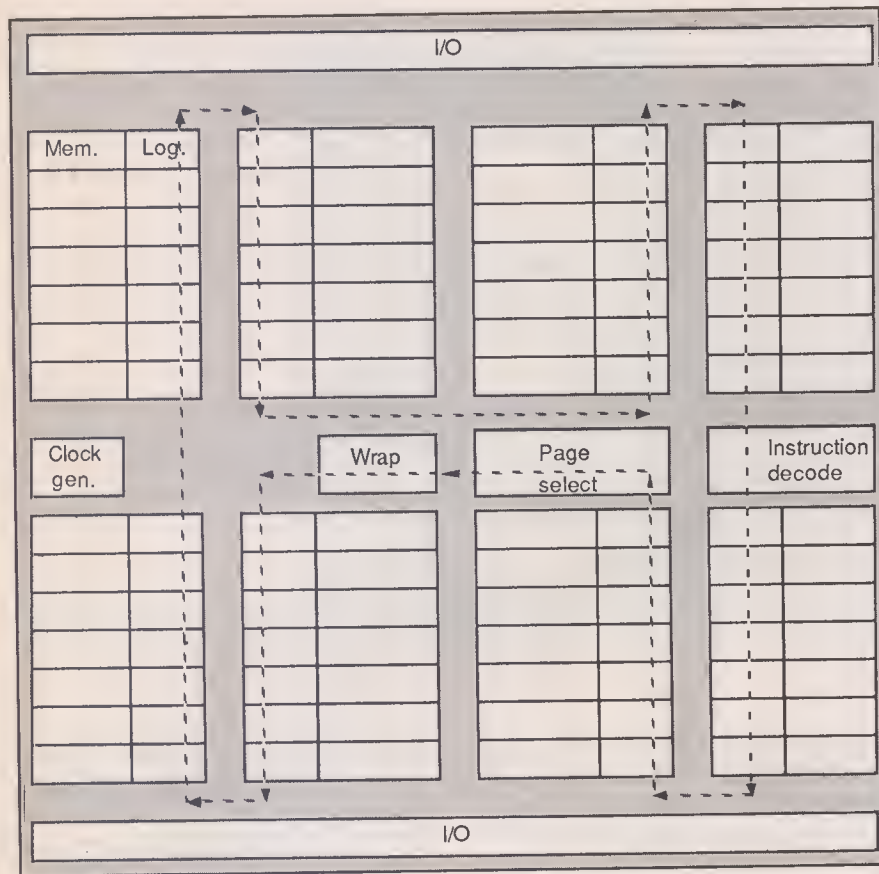


Figure 5. PAM floor plan.

are stored for every "match" section of logic. In effect, this logic is multiplexed over the storage so as to achieve a workable memory density.

The logic part consists of comparators for the name fields and small finite-state machines operating on the tags to perform the matching function. Fifty-six of these blocks are included on each PAM chip, as shown in Figure 5.

The chip contains 560 words of 16-bit symbols. If an average clause head has a size of seven PAM words, one chip can store 80 such expressions. The PAM chip is designed so that the controller can run a number of them together and in parallel without the need for "glue" logic. In this way, the total capacity of the PAM is only limited by the number of chips used. Owing to its parallel operation, the speed of the matching function is independent of the PAM's size.

We have presented an overview of the physical architecture of a novel, high-performance, symbolic multiprocessing system known as FAIM-1. A small-scale prototype of

FAIM-1 that consists of a single *E-3* surface (19 PEs) is under construction. The primary performance mechanism is the exploitation of concurrency at all levels of the system. From the top, the FAIM-1 appears as a medium-grain, homogeneous multiprocessor. The topology scales to permit an arbitrary number of processing elements to be interconnected. Each processing element can be viewed from the bottom as a heterogeneous, shared-memory multiprocessor containing servers that are specialized to perform message delivery, mock-unification, data storage and delivery, instruction storage and delivery, instruction evaluation, and rapid context switching.

The architecture takes advantage of advanced circuit and packaging technology as a secondary performance enhancement mechanism. The present small-scale prototype is being implemented with custom CMOS VLSI circuits (the Post Office, PAM, and ISM subsystems) and commercially available components (the processor and SRAM). The 19-PE initial prototype will be attached to a host Symbolics Lisp Machine. The programming environment, OIL compiler, resource allocator,

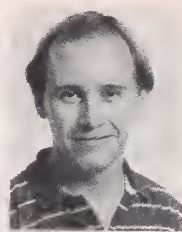
debugging tools, and a complete system simulator that can be used for initial application development all run on the Host machine. All of these characteristics will enable the FAIM-1 prototype to serve as a symbolic accelerator to the host Lisp Machine.

While the architecture represents a novel and significant reallocation of the conventional transistor budget so as to provide high performance and efficient evaluation of highly concurrent symbolic programs, the programming environment departs from conventional AI program development systems only minimally so as to incorporate concurrent application development. This eliminates the need to re-educate programmers, and it means that the architecture can be scaled to provide two to three orders of magnitude performance improvement over conventional AI machines. □

References

1. C. L. Seitz, *Introduction to VLSI Systems*, "System Timing" (Chapter 7), McGraw-Hill, New York, 1979.
2. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag, New York, 1981.
3. J. A. Rees, N. I. Adams, and J. R. Meehan, "The T Manual," technical report, 4th ed., Yale University, Computer Science Dept., New Haven, Conn., 1984.
4. J. S. Conery and D. F. Kibler, "Parallel Interpretation of Logic Programs," in *Proc. Conf. Functional Programming Languages and Computer Architectures*, ACM, Oct. 1981, pp. 163-170.
5. D. DeGroot, "Restricted AND-Parallelism," *Proc. Int'l Conf. on Fifth-Generation Computer Systems*, Nov. 1984, pp. 471-478.
6. C. L. Seitz, "The Cosmic Cube," *Comm. ACM*, Vol. 28, No. 1, Jan. 1984, pp. 22-33.
7. R. Gurwitz, "The Butterfly Multiprocessor," talk presented at the 1984 ACM National Convention, San Francisco, Oct. 1984; also, "Butterfly Parallel Processor Overview," BBN Laboratories, Inc., 10 Moulton St., Cambridge, MA 02238, Dec. 18, 1985.
8. S. J. Stolfo, D. Miranker, and D. E. Shaw, "Architecture and Applications of DADO, A Large-Scale Parallel Computer for Artificial Intelligence," *Proc. Eighth Int'l Joint Conf. Artificial Intelligence*, Karlsruhe, West Germany, Aug. 1983, pp. 850-854.
9. D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, Mass., 1982.
10. D. Gordon, I. Koren, and G. M. Silberman, "Fault-Tolerance in VLSI Hexagonal Arrays," technical report, Dept. of Electrical Engineering, Computer Science Technion, Haifa 32000, Israel, 1984.
11. John L. Hennessy, "VLSI Processor Architecture," *IEEE Trans. Computers*, Vol. C-33, No. 12, Dec. 1984, pp. 1221-1246.
12. B. W. Lampson, G. McDaniel, and S. M. Ornstein, "An Instruction Fetch Unit for a High-Performance Personal Computer," *IEEE Trans. Computers*, Vol. C-33, No. 8, Aug. 1984, pp. 712-730.

13. K. S. Stevens, S. V. Robison, and A. L. Davis, "The Post Office—Communication Support for Distributed Ensemble Architectures," *Proc. Sixth Int'l Conf. Distributed Computing Systems*, May 1986, pp. 160-167.
14. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Vol. 3, 1979, pp. 267-286.
15. D. Warren, "Implementing Prolog," Technical Report 39, Edinburgh University, May 1977.



William S. Coates joined the AI Architectures Group at Schlumberger Palo Alto Research in 1984. He received his BS in computer science in 1980, and an MS in computer science in 1985, both from the University of Calgary, Alberta, Canada.



Ian N. Robinson is a member of the AI Architectures Group at Schlumberger's Palo Alto Research Center. He received a BSc degree in physics with electronics from the University of Sussex, England, in 1979.

Selected bibliography

Davis, A. L., and S. V. Robison, "The Architecture of the FAIM-1 Symbolic Multiprocessing System," *Proc. Ninth Int'l Joint Conf. Artificial Intelligence*, Aug. 1985, pp. 32-38.

Goldberg, A., and D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, Mass., May 1983.

Kluge, W. E., and K. Lautenbach, "The Orderly Resolution of Memory Access Conflicts Among Competing Channel Processes," *IEEE Trans. Computers*, Vol C-31, No. 3, Mar. 1982, pp. 194-207.

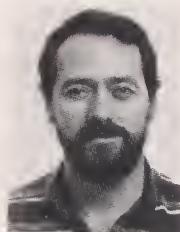
Lee, J. K. F., and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, Vol. 17, No. 1, Jan. 1984, pp. 6-22.

Lyon, R. F., "Single Instruction Stream, Multiple Data Stream, Multiple Port Pipelined Processor," Schlumberger internal report, Schlumberger Palo Alto Research, 3340 Hillview Ave., Palo Alto, CA 94304, Sept. 1985.

Martin, A. J., "The Torus: An Exercise in Constructing a Surface," *Proc. Second Caltech Conf. VLSI*, Jan. 1981, pp. 527-538.

Przybylski, S. A., et al., "Organization and VLSI Implementation of MIPS," *J. VLSI and Computing Systems*, Vol. 1, No. 3, Fall 1984, pp. 170-208.

Transputer INMOS Limited, "IMS T424 Transputer Reference Manual," Whitefriars, Lewins Mead, Bristol, England, UK, 1984.



A. L. Davis received a BS in electrical engineering from MIT in 1969, and a PhD in electrical engineering and computer science from the University of Utah in 1972. He is currently a member of the AI Architectures Group at Schlumberger Palo Alto Research. He is a member of IEEE.



Shane V. Robison is a member of the AI Architectures Group at Schlumberger's Palo Alto Research Center. He received his BS and MS degrees in computer science from the University of Utah, Salt Lake City, Utah, in 1980 and 1983, respectively. He is a member of IEEE, ACM, and AAAI.



Judy Anderson is a member of the research staff of the AI Architectures Group at Schlumberger Palo Alto Research. She received a BA in philosophy in 1983 and an MS in computer science in 1984, both from Stanford University.



Robert W. Hon is a member of the AI Architectures Group at Schlumberger Palo Alto Research. Prior to joining Schlumberger, he was an assistant professor of computer science at Columbia University. He holds a BS in electrical engineering from Yale University and an MS and PhD in computer science from Carnegie Mellon University. He is a member of IEEE and ACM.



Kenneth S. Stevens is a member of the research staff of the AI Architectures Group at Schlumberger Palo Alto Research. He received a BA degree in biology in 1982, and BS and MS degrees in computer science in 1982 and 1984 from the University of Utah. He is a member of IEEE.

Readers may write to Shane V. Robison at Schlumberger Palo Alto Research, 3340 Hillview Ave., Palo Alto, CA 94304.



WESTEX-87

The Second Western Expert Systems Conference

Disneyland Hotel, Anaheim, California

June 2-4, 1987

Sponsored by the Western Committee of the
Computer Society of the IEEE and the
IEEE Los Angeles Council



CONFERENCE SCHEDULE

Paper Deadline:

February 1, 1987

Acceptance Decisions Mailed:

March 1, 1987

Camera-ready Copies Received:

March 27, 1987

Advance Registration Closes:

April 30, 1987

Tutorials:

June 2, 1987

Conference Sessions:

June 3, 4 1987

For further information, please
contact:

Mr. Leonard Beck—General Conference Information

Hughes Aircraft Company
P.O. Box 2111
Fullerton, CA 92633-0111
(714) 732-4092

Mr. Keith Warn—Papers

Rockwell International
(714) 779-3174

Mr. Jeffrey Pifer—Exhibits

Unocal
(714) 528-7201

Dr. Eugene Hostetter

UC Irvine
(714) 856-5491

ADVANCE REGISTRATION

Deadline: April 30

Non-IEEE Members

June 2 Tutorials—\$160
June 3, 4 Sessions—\$175

IEEE Members

June 2 Tutorials—\$115
June 3, 4 Sessions—\$140

Expert systems are a new form of computer application that has recently received attention because of their success in automating and multiplying the ability of human experts in diverse fields. The WESTEX-87 conference will focus on the practical use of expert systems. The conference will include sessions and panels on examples of successful systems, methods for developing expert systems and tools for developing expert systems.

The conference will be held at the Disneyland Hotel in Anaheim, California. In addition to the conference, the Hotel boasts the proximity of Disneyland, Knott's Berry Farm, and other Southern California attractions. A block of rooms has been set aside for conference attendees.

Conference Attractions:

1. The conference program consists of two days of sessions including papers and panels on examples of successful expert systems, methods for developing expert systems, and special sessions on hybrid systems and diagnostic tools.
2. Invited speakers will address the future potential of expert systems and their cultural impact.
3. Parallel tutorial tracks will cover beginning and advanced expert-system development. The beginner track will cover "getting started in expert systems." The advanced track will cover tools and methodologies in detail.
4. Exhibits will show the latest in support tools and expert-system development environments.

To register in advance, clip and mail to WESTEX-87
P.O. Box 2111, Fullerton, CA 92633-0111

Name _____

Company _____

Address _____

City _____

State, Zip _____

☐ Conference Tutorials

☐ Exhibits Only

☐ Conference Sessions

☐ Send Details

IEEE Membership # _____

M2 _____

What Price Smalltalk?

David Ungar, Stanford University

David Patterson, University of California, Berkeley

Are reduced instruction set computers good for languages like Lisp and Smalltalk? With the addition of a few simple hardware changes and software techniques—yes.

In anticipation of the promise of tremendous hardware advances, software researchers have fashioned expansive programming environments to improve programmer productivity. Even with the march of technology, exploratory programming environments such as Smalltalk-80^{1,2} require such expensive computers that few programmers can afford them. With the hope of increasing that community, a research project at the University of California created a reduced instruction set computer³ for Smalltalk on a RISC.^{4,5} Figure 1 shows SOAR, a 32-bit NMOS microprocessor.

We are now able to estimate the performance implications of the Smalltalk-80 programming environment. In the next section we list the demands that Smalltalk-80 places on traditional computer systems, then present software and hardware ideas that answer those demands.

We have learned a number of lessons from the SOAR microprocessor, including the hard lesson that we now call "the architect's trap."

The demands of Smalltalk-80

To improve programmer productivity, Smalltalk-80 removes four restrictions found in conventional programming systems. First, Smalltalk-80 programmers do not have to declare the types of variables at compile-time.

(1) Run-time data typing instead of compile-time type declarations. The Fortran statement "I = J + K" denotes integer addition and can be performed with a single Add instruction. But Smalltalk-80 has no type declarations, so J and K may hold values of any type, from Booleans to B-trees. Thus, every time a Smalltalk-80

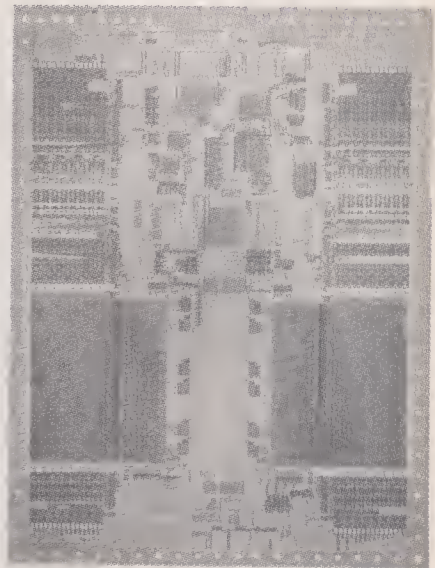


Figure 1. Microphotograph of the SOAR microprocessor. Using 4-micron line widths, Joan Pendleton led the implementation of this 32-bit microprocessor.^{6,7} It is 10.7 mm by 8.0 mm, uses 35,700 transistors, dissipates about 3W, and runs about 400 ns per instruction. These chips were fabricated by MOSIS, and have performed better than expected. Xerox also volunteered to make some SOAR chips. Although they run at 330 ns per instruction, the Xerox chips do not correctly perform all tests. We are considering a fabrication attempt with 3-micron line widths to further reduce the cycle time.

system evaluates "J + K" it must first check the types and then perform the appropriate operation.

Measurements of conventional Smalltalk-80 systems show that over 90 percent of the "+" operations do the simplest possible operation, integer addition.⁸ Since a type check takes at least as long as an Add instruction, most Smalltalk-80

Table 1. Speed of Smalltalk-80 systems.

System	Maker	Host Processor	Instruction Time in ns	Execution Model	Speed
BS	UCB	68010	400 (18%)	interpreter	11%
Tek 4404	Tektronix	68010	400 (18%)	interpreter	25%
Dorado	Xerox	Dorado	67 (100%)	microcode	100%
SOAR	UCB	SOAR	400 (18%)	compiler	107%

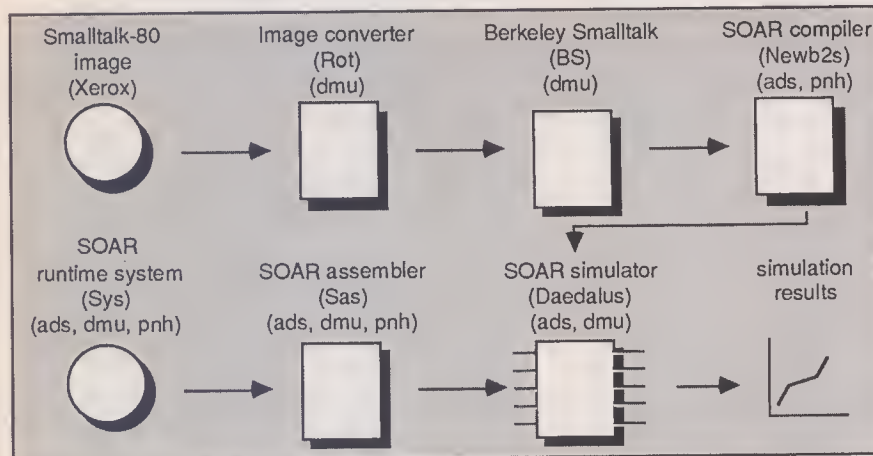


Figure 2. Steps involved in a SOAR simulation. First, Rot removes the object table from the Xerox Smalltalk-80 image. We then use BS to make any modifications necessary in the image (to eliminate some becomes). Newb2s produces a Smalltalk image for SOAR by converting the BS objects to SOAR format, and Hilfinger's Slapdash compiler translates the bytecoded programs into SOAR instructions. We have also coded the Smalltalk primitive operations and storage management software in SOAR assembly language. After this is assembled, it is fed to Daedalus, our SOAR simulator, along with the Smalltalk image. The initials below each system show its author: ads is Dain Samples, pnh is Paul Hilfinger, and dmu is David Ungar.

systems waste a lot of time checking types for integer arithmetic.

(2) Dynamically bound messages instead of statically bound procedures. Smalltalk was probably the first system to be called *object oriented*, an approach to programming that associates routines with data structures to encourage programmers to think of programming as sending messages to data objects rather than as calling procedures to update shared data. Thus, the routine to be invoked depends on the type of data, and since the type of data can change at run-time, the equivalent of a simple procedure call on most systems is a comparison and table lookup in Smalltalk-80.

(3) Fast compilation to interpreter instead of optimizing compilation to native hardware. A key to improving programmer productivity is to reduce the time it takes to change a program and then test it.

To shorten the edit-compile-debug cycle, the Smalltalk programmer recompiles a routine using a simple and fast compiler that produces code for a Smalltalk interpreter. The Smalltalk interpreter provides the foundation for an excellent debugger, which further improves programmer productivity.

Like many other systems, Smalltalk supports separate compilation so that the programmer only waits for one routine to be recompiled rather than the whole system. But unlike other systems, Smalltalk further shortens this cycle by avoiding the conventional linking step: routines are linked "on the fly" as needed. To shorten program development, Smalltalk sacrifices potential compile time optimizations and static linking that lead to faster program execution.

(4) Automatic dynamic storage allocation and reclamation instead of program-

mer-controlled dynamic structures. The designers of Smalltalk believed that programmers should not be encumbered with managing dynamically varying data, so storage management was left to the system. Exploratory programming environments have the added constraint of avoiding distracting pauses while collecting garbage, since a major contributor to programmer productivity is a fast and predictable interaction with the programming environment.⁹ Smalltalk is even more challenging for garbage collection, since Smalltalk programs tend to generate garbage about 10 times faster than most Lisp programs.

The costs on traditional systems

Given the demands mentioned above, the obvious question is, how well does Smalltalk-80 run on conventional computer systems? The first step in answering this question is determining a fair way to measure performance. It is always dangerous to rely on architects to supply the tests of their creations, and fortunately we were not put in that predicament. The creators of Smalltalk have a standard set of benchmarks that all Smalltalk implementors use to compare systems.¹⁰ Xerox rates performance by taking the mean of 13 macro-benchmarks, plus the text scanning and BitBlt micro-benchmarks. Macro-benchmarks include compilation, decompilation, searching the Smalltalk program hierarchy, and so on.

Table 1 compares the performance on several Smalltalk-80 systems using five of those large benchmarks. Both the Tek 4404 and Berkeley Smalltalk (BS) use software interpreters, with BS being the experimental vehicle that we used to study Smalltalk. The Dorado is the machine used at Xerox to run Smalltalk. The SOAR chip has been placed on a board, the board has been placed in a workstation, and this package is running diagnostic programs, but we have not brought up the complete Smalltalk system, so we must use simulations to estimate performance (see Figure 2). This system was described at a recent conference.¹¹

The system generally agreed to be fast enough for Smalltalk is the Dorado, a \$120,000 ECL* personal computer that supports a single programmer at a time. (Think of it as a personal computer for

*Emitter-coupled logic is an expensive technology used in mainframes and super-computers.

people who consider the Mercedes Benz a commuter car.) If we compare the Dorado to the Smalltalk interpreters on Motorola 68010 microprocessor-based systems, we see that the Dorado is four to eight times faster, which might be expected since the instruction time of the Dorado is about six times faster than the instruction time of the 68010. This is not the case for SOAR, which is about the same speed as the Dorado even though the instructions are six times slower. SOAR is faster than expected because it uses several innovations in software and hardware (see Table 2), as explained in the next two sections.

Table 2. Summary of innovations.

ST-80 Demands	Category	Name	% slower if omitted
interpretation	Software	compiler	100%
dynamic messages		in-line cache	33%
		direct pointers	20%
automatic storage		generation scavenging	12%
slow message sends	Architecture	register windows	46%
		tagged integers	26%
run-time data types		two-tone instructions	16%

Reducing costs through software innovation

The first place to look for performance improvements is with software; it will be a very long time before it is as easy to build and debug hardware as it is to write and debug the equivalent program. Architects should only consider hardware implementations when there are tremendous gains to be had in total system performance, and when they are sure that function will never change. As we shall see, architects don't always follow this sound advice.

Interpretation. The Smalltalk-80 system is defined by a stack-oriented virtual machine based on the Dorado Smalltalk-80 implementation.¹² Each instruction consists of one to three bytes and generally corresponds to a token in the source program. These instructions are usually called *bytecodes*. Bytecodes have the following advantages:

- The simple correspondence between source and object code simplifies the compiler and debugger.

- Smalltalk can be transported to a new machine by writing only the virtual machine emulator.

This approach has drawbacks too:

- Decoding such dense instructions takes either substantial hardware or substantial time. For example, the instruction fetch unit consumes 20 percent of the Dorado CPU,¹³ and decoding a simple bytecode takes twice as long as executing it in Berkeley Smalltalk.

- Some of the high-level instructions require many microcycles to execute. These multicycle instructions must be sequenced by a dedicated control unit.

The alternative is simply to compile to the native instruction set of the machine, hoping to gain performance with the traditional advantage of compilation over in-

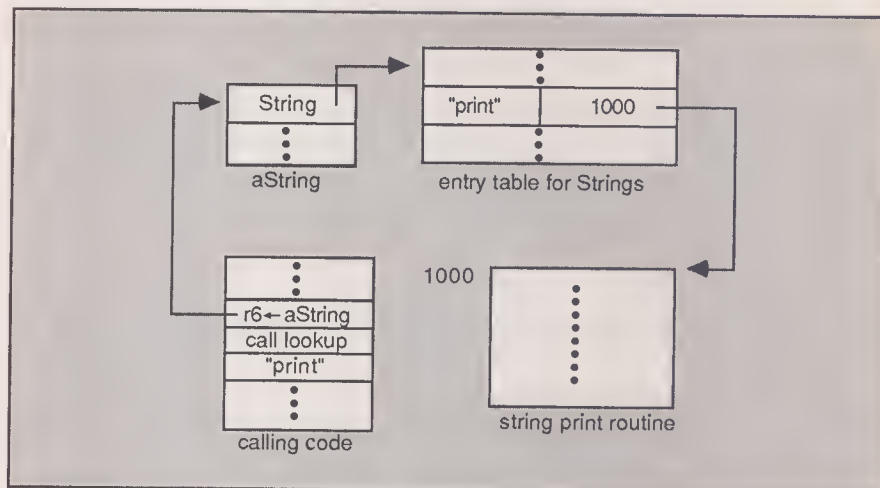


Figure 3. Caching the target address in the instruction stream. In this example, the print routine is called with an argument that is a string. (The argument is passed in r6.) The first time the call instruction is executed, the call contains the address of a lookup routine and the word after the call contains a pointer to the name "print." The lookup routine follows the pointers to the entry table for strings, and finds the entry for "print." It then overwrites the call instruction with a call to that routine and replaces the word after the call with the type of the argument (string). (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," 11th An. Int'l Symp. Computer Architecture, © IEEE.)

terpretation. We estimate compilation improves the performance of SOAR by 100 percent. A negative consequence of our decision to abandon bytecodes is that it forces us to rewrite the Smalltalk-80 debugger. Lee has designed a debugger for SOAR and has built a prototype in Berkeley Smalltalk.¹⁴ He has exploited the hardware organization of SOAR in the design of the debugger to add a conditional breakpoint facility and increase execution speed during debugging.

SOAR may also have played a role in inspiring a compiler project at Xerox by Deutsch and Schiffman.¹⁵ This implementation, called PS, provided the speed of compilation but maintained consistency with the interpreter so that the debug-

ger would not have to be modified. Their system runs almost twice as fast as interpreters using the same microprocessor.

Caching call targets in-line. Another way to improve Smalltalk performance is by decreasing the time taken to find the target of a call. Once computed, the target's address can be cached in the instruction stream for later use, as suggested by Schiffman and Deutsch.¹⁵ Figures 3 and 4 illustrate this idea. This in-line caching exacts a price for its time savings: the processor must support nonreentrant code.

Although complicated, in-line caching pays handsome rewards. The conventional way for Smalltalk systems to save call

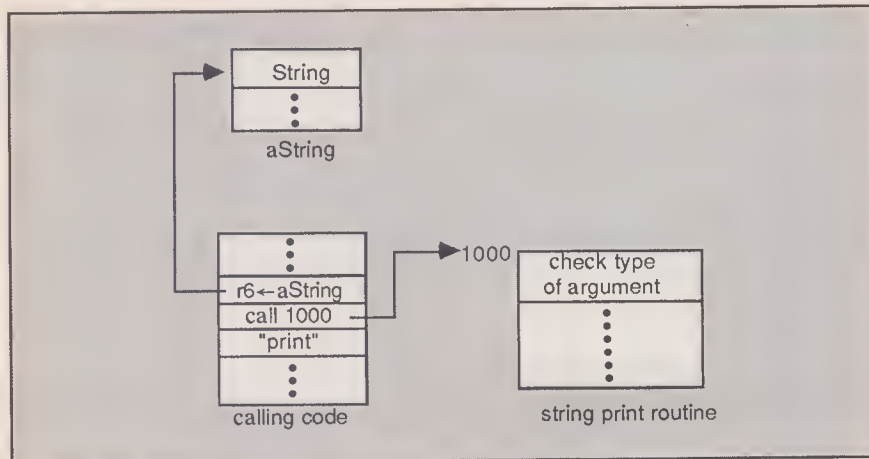


Figure 4. Caching the target address in the instruction stream. The next time the call is executed, control goes directly to the string print routine. A prolog checks that the current argument's type matches the contents of the word following the call instruction. This word contains the type that the argument had the previous time the call was executed. If the types match, control falls through to the string print routine, otherwise another table lookup is needed. (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," *11th An. Symp. Computer Architecture*, © IEEE.)

Table 3. Summary of generation scavenging's performance.

	Berkeley Smalltalk	SOAR
execution model	interpreted	compiled
source of data	measurements	simulations
processor	MC68010	SOAR
cycle time	400 ns	400 ns
CPU time overhead		
mean	1.5%	0.9%
worst case	N.A.	3.3%
pause time (scavenge duration)		
mean	160 ms	19 ms
worst case	330 ms	28 ms
peak main memory usage	200 KB	200 KB

targets is a hash table. But the overhead for probing into a hash table would slow SOAR by 33 percent. The penalty for in-line caching is a software trap mechanism to synchronize process switches and cache probes. If we were forced to omit this, we could use a table containing the last addresses with an indirect in-line cache. This would slow SOAR down by 7 percent. Even with in-line caching, SOAR still spends 11 percent of its time in call-target cache probes and another 12 percent handling misses. Further research into computing the call's target might yield even more substantial savings.

Object-oriented storage management.

Software is also the architect's best option for managing dynamically varying data structures, called objects in Smalltalk-80. Smalltalk-80 objects are smaller and more volatile than data structures in most other exploratory programming environments, averaging 14 words in length and living for about 500 instructions. Smalltalk-80 systems face three challenges in managing storage for objects:

- Automatic storage reclamation. On average, 12 words of data are freed and must be reclaimed per 100 Smalltalk-80 virtual machine bytecodes executed.

- Virtual memory. All objects must be in the same address space.

- Object-relative addressing. Although offsets into objects are known at compile-time, base addresses are not. Code must be compiled to address fields relative to dynamically determined base addresses.

Automatic storage reclamation. SOAR supports generation scavenging¹⁶ to reclaim storage efficiently without requiring costly indirection or reference counting. This algorithm is based on the observation that most objects either die young or live forever. Thus, objects are placed into two generations and only new objects are reclaimed.

Storage reclamation has a strong impact on performance; most other algorithms would squander 10 percent to 15 percent of SOAR's time on automatic storage reclamation instead of generation scavenging's 3 percent (see Table 3). Hence, without generation scavenging SOAR would take 4 percent to 15 percent more cycles to run the benchmarks.

Traditional software and microcode implementations of object-oriented systems rely on an object address table (see Figure 5). Each word of an object contains an index into this table, and the table entry contains the address of each object. The indirection through the table primarily supports compaction. Generation scavenging provides compaction for free, permitting SOAR to function without an object table (see Figure 6). Without this algorithm, the extra work to follow indirect pointers through the object table would slow SOAR down by 20 percent. These results seem confirmed by a recent Smalltalk-80 interpreter that used generation scavenging with direct pointers.¹⁷ Although we would expect it to be a factor of two slower than PS because it is interpreted rather than compiled, this implementation is close to 90 percent of the speed of PS. It may be that direct pointers and generation scavenging, not used in PS, explain the surprisingly close performance.

Reducing costs through architectural innovation

The basic theme of the SOAR architectural additions is to allow the normal case to run fast in hardware and to trap to software in the infrequent complicated case.

Tags trap bad guesses. SOAR follows the Smalltalk-80 virtual machine in sup-

porting only two data types with tags: integers and pointers.¹ But SOAR departs from the Smalltalk-80 virtual machine by starting arithmetic and comparison operations immediately and simultaneously checking the tags. (Figure 7 shows the SOAR tag layout.) Most often both operands are integers and the correct result is available after one cycle. If not, SOAR aborts the operation and traps to routines that carry out the appropriate computation for the data types. SOAR is the only Smalltalk-80 system that overlaps these operations. Without tagging, SOAR would run 26 percent slower.

A tagged architecture that lacks micro-code must include instructions that manipulate and inspect tags. Because the Smalltalk system already relies on the compiler to ensure system integrity, we can allow the compiler to mix instructions that manipulate tags with instructions constrained by tags.

Each SOAR instruction contains a bit that either enables or disables tag checking. The untagged mode turns off all tag checking and operates on raw 32-bit data. In untagged mode the tag bits are treated as data, and the complete instruction set can be used to manipulate this data. Untagged instructions also allow programs written in conventional languages such as C and Pascal to run on SOAR. Instead of providing two versions of each instruction, we could have defined a mode bit in the program status word (PSW). This would have been expensive, increasing execution time by 16 percent.

Multiple overlapping on-chip register windows. We used hardware to improve the performance of arithmetic operations, and also to improve the performance of the frequent calls and returns. SOAR, like RISC I, optimizes them by providing a large, on-chip register file. The registers are divided up into overlapping windows. Instead of saving or restoring registers, calls or returns merely switch windows (Figure 8).

When we compared Smalltalk to the C language subroutines, we found that the shorter Smalltalk subroutines pass fewer operands and use fewer local variables, and so need fewer registers. So each SOAR register window has 8 registers instead of 12 for RISC I. Figures 8 and 9 show the register organization of SOAR. In addition to 56 more registers, the inclusion of register windows results in the addition of a register to select the current window, a register to detect overflows by

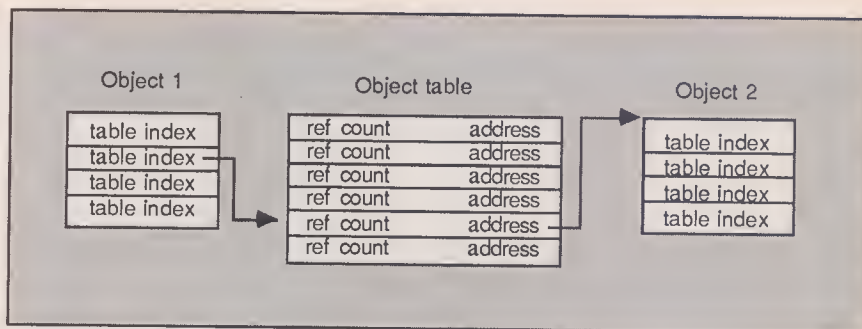


Figure 5. Indirect addressing. In traditional Smalltalk-80 systems, each pointer is really a table index. The table entry contains the target's reference count and memory address. This indirection required previous Smalltalk-80 systems to dedicate base registers to frequently accessed objects. The overhead to update these registers slowed each procedure call and return. (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," *11th An. Symp. Computer Architecture*, © IEEE.)

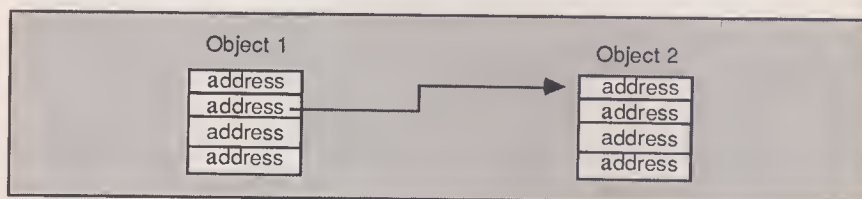


Figure 6. Direct addressing. A SOAR pointer contains the virtual address of the target object. This is the fastest way to follow pointers. (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," *11th An. Symp. Computer Architecture*, © IEEE.)

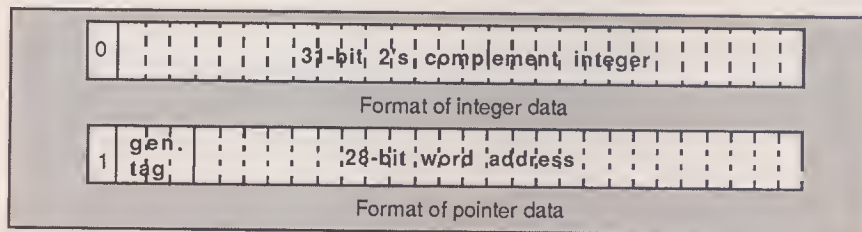


Figure 7. SOAR tagged data types. SOAR supports two data types, 31-bit signed integers and 28-bit pointers. Pointers include a generation tag. SOAR words could have contained 32 bits of data plus one bit of tag for a total of 33 bits. The scarcity of 33-bit tape drives, disk drives, and memory boards led us to shorten our words to a total of 32 bits including the tag (31 bits of data). (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," *11th An. Symp. Computer Architecture*, © IEEE.)

recording the last saved window, more elaborate register decoders, and trapping logic.⁷

Despite the cost of all the added hardware, Smalltalk-80's predilection for procedure calls makes the register windows very important. The cost of saving and restoring a conventional register file would slow the machine down by 46 percent, even with Load- and Store-multiple instructions, which we discuss below.

Register windows allow many registers per procedure without making procedure calls and returns expensive.³ Many RISC

machines use optimizing compilers to cleverly allocate variables to registers and thus avoid register windows. While this may be the right way to go with conventional programming systems like C and Pascal, the extra compile time of register allocation may be inappropriate for exploratory programming environments such as Smalltalk.

Architect's trap

After the careful and clean design of the first Berkeley RISC projects, SOAR went

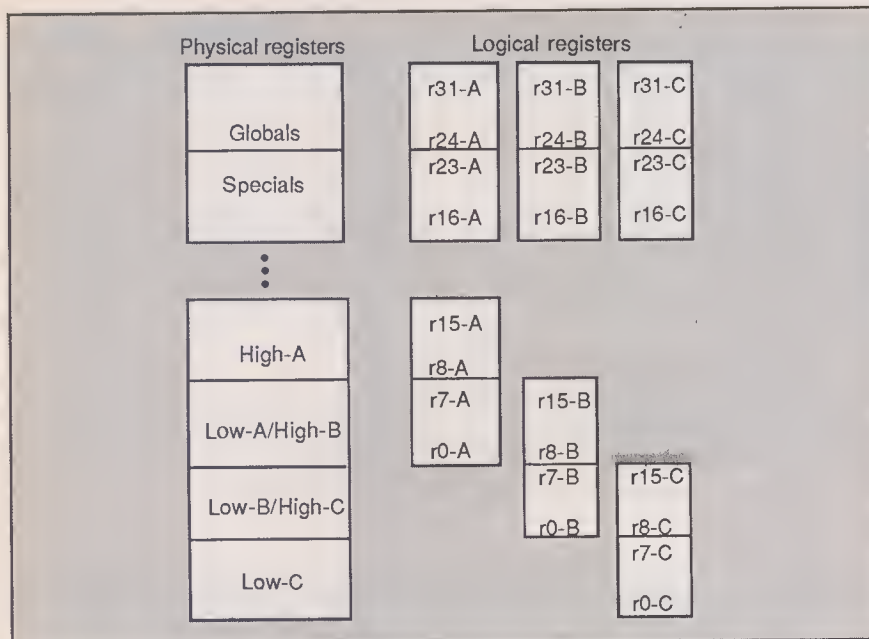


Figure 8. SOAR's register windows. Like RISC I, SOAR has many physical sets of registers that map to the logical registers seen by each subroutine. (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," *11th An. Symp. Computer Architecture*, © IEEE.)

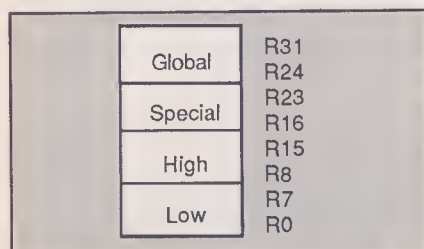


Figure 9. Logical view of the register file. The Highs hold incoming parameters and local variables. The Lows are for outgoing arguments. The Specials include the PSW and a register that always contains zero. The Globals are for system software such as trap handlers. (Reprinted from "Architecture of SOAR: Smalltalk on a RISC," *11th An. Symp. Computer Architecture*, © IEEE.)

on to try many architectural ideas. Although fully aware of the dangers, we fell into what we now call the "architect's trap" several times:

- Each idea was a clever idea;
- each idea made a particular operation much faster;
- each idea increased design and simulation time; and
- not one idea significantly improved overall performance!

Put another way, the primary impact of these clever ideas was to increase the difficulty of building SOAR and thus lengthen the development cycle.

One example is Load- and Store-multiple mentioned above. When the number of activations on the stack exceeds the on-chip register capacity, SOAR traps to a software routine that saves the contents of a set of registers in memory. Unlike RISC II, SOAR has load- and store-multiple instructions to speed register saving and restoring. These instructions can transfer eight registers in nine cycles (one instruction fetch and eight data accesses). Without them, the system would need eight individual instructions that would consume sixteen cycles (eight instruction fetches plus eight data accesses). In retrospect, these multicycle instructions added some complexity to the design, and the benefit of a three percent reduction of execution time is not worth the costs.

Another example comes from generation scavenging, which requires that a list be updated whenever a pointer to a new object is stored in an old object. While designing SOAR, we thought that stores would be frequent enough to warrant hardware support for this check. Thus, SOAR tags each pointer with its generation of the object that it points to. While computing the memory address, the store

instruction compares the generation tag of the data being stored with the generation tag of the memory address. For 96 percent of the stores, list update is unnecessary and the store completes without trapping. Once again we rely on tags to confirm the normal case and trap in the unusual case. Surprisingly, tagged stores are so infrequent that hardware support saves only 1 percent of the total time. This feature does not seem worth the effort.

Alas, we have more than two examples of being caught in the architect's trap. We have put these observations to use by calculating the performance of some variations on SOAR and comparing them to some real systems (see Table 4). Our predictions of SOAR's performance are based on simulated macro-benchmark times and do not include virtual memory, operating system, and I/O overhead. However, all of the Smalltalk-80 systems we know about tend to be limited by the raw CPU speed rather than the overhead time for virtual memory, the operating system, and I/O. For a fair comparison, we assume a 400-ns cycle time for SOAR, RISC II, and MC68010.

By comparing the speeds of different systems, we can gain some insight into the reasons for SOAR's good performance.

• If we were to build a second-generation SOAR, we would implement Pendleton's reorganization of the pipeline, which would shorten SOAR's cycle time by 25 percent.⁷ By combining it with the winning features from the previous table the system would run faster than SOAR, even though it lacks a half-dozen architectural innovations that SOAR has. While the overall measurements confirm that reduced instructions sets can run Smalltalk-80 efficiently, this calculation goes further; it suggests that SOAR is not RISCy enough.

• The software techniques developed for SOAR may eventually be used to construct fast, compiled Smalltalk-80 systems on general-purpose processors. The RISC II row of the table reflects the impact of removing the Smalltalk-specific architectural features of SOAR. The next row, RISC II without register windows, is representative of the commercial reduced-instruction-set microprocessors. These machines would run Smalltalk-80 twice as slow as SOAR. This may not be too high a price to pay for a simpler and more general off-the-shelf design.

Falling into architect's traps will delay your project, making it less attractive as

other, competing, machines are announced. For example, the Deutsch-Schiffman Smalltalk system PS doubled performance when Motorola announced their 32-bit follow-on to the 68010, called the 68020. It includes a small on-board instruction cache, executes instructions in 180 ns, and uses about five times as many transistors as SOAR. Thus, delays in your project inevitably reduce the relative attractiveness of your machine, since it is unlikely that every other project will face the same delays.

Table 2 summarized the innovations used in SOAR to improve the performance of the exploratory programming environment Smalltalk-80. If we compare SOAR to the Motorola 68010, we have more than doubled performance while halving the number of transistors. Its features reduce the performance penalty for Smalltalk from a factor of 3.6 to a factor of 1.3, increasing the chances that programmers will use exploratory programming environments (see Table 5).

We recommend that anyone faced with the task of building a computer for an exploratory programming environment consider compilation to a reduced instruction set with register windows and one-bit tags. There are four places to look for further performance gains: compiler technology, implementation technology,⁷ optimization of the run-time support primitives (which consume about two-thirds of SOAR's time), and better hardware or software algorithms to cache call-target lookups (which consume 23 percent of SOAR's time). Of these, implementation technology—circuit design and VLSI processing technology—have the most dramatic impact. Since we started this project, the standard VLSI technology available to universities has improved from four-micron line widths to three-micron. This one change could reduce our cycle time from 400 ns to 290 ns, as important a contribution as register windows.

We also warn all architects to watch their step. Be sure you evaluate your proposal in terms of benefits to the whole system and not just to one part, and remember that a software solution is better than a hardware solution in every facet except speed. The bait of the architect's trap is your own ingenuity, but the trap ensnares a whole project, not just the fool who springs it. Beware! □

Table 4. Trimming the fat from SOAR.

Assumes 400-ns cycle time for SOAR, RISC II, and 68010. Dorado and MC68010 figures are measurements. Same benchmarks as Table 1.

Configuration	Speed
Important architectural features + rearranged pipeline	103%
full SOAR	100%
Dorado	93%
important architectural features only	81%
Deutsch-Schiffman 68020	72%
RISC II	62%
RISC II without register windows	50%
full SOAR without software ideas	41%
Deutsch-Schiffman 68010	36%
full SOAR without hardware ideas	34%
Tek 68010 interpreter	24%
stripped SOAR	22%

Table 5. Cost of Smalltalk-80 features: the standard approach versus SOAR.

ST-80 Feature	Cost (% slower)	Optimization	Cost (% slower)
interpretation	100%	compiler	0%
dynamic messages	56%	in-line cache	23%
automatic storage	15%	generation scavenging	3%
object table	20%	direct pointers	0%
run-time data types	26%	tagged integers	0%
many subroutines	46%	register windows	0%
total cost of ST-80 (standard architecture)	3.6:1	(SOAR)	1.3:1

Acknowledgments

The SOAR project was sponsored by DARPA, and many students and faculty participated. We thank all of them for their hard work. We also wish to thank Walter Beach and Garth Gibson for making comments on an early draft of this article.

References

1. Adele J. Goldberg and David Robson, *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, Reading, MA, 1983.
2. Ted Kaehler and Dave Patterson, *A Taste of Smalltalk*, W. W. Norton and Company, New York, NY, 1986.
3. D. A. Patterson, "Reduced Instruction Set Computers," *Comm. ACM*, Vol. 28, No. 1, Jan. 1985, pp. 8-21. Special issue on computer architecture.
4. David Ungar et al., "Architecture of SOAR: Smalltalk on a RISC," *11th An. Int'l Symp. Computer Architecture*, Ann Arbor, MI, June 1984, pp. 188-197.
5. David M. Ungar, "The Design and Evaluation of a High Performance Smalltalk," C. S. tech. report no. UCB/CSD 86/287 (PhD dissertation), Computer Science Division (EECS), Univ. of California, Berkeley, CA, Mar. 1986. Also to be published by MIT Press as part of the ACM Doctoral Dissertation Award series.
6. J. M. Pendleton et al., "A 32b Microprocessor for Smalltalk," *IEEE J. Solid State Circuits*, Oct. 1986.
7. Joan Pendleton, "A Design Methodology for VLSI Processors," PhD dissertation, Dept. of EECS, Univ. of California, Berkeley, CA, Sept. 1985.
8. Ricki Blau, "Tags and Traps for the SOAR Architecture," *Smalltalk on a RISC: Architectural Investigation*, ed. David A. Patterson, Computer Science Div., Univ. of California, Berkeley, CA, Apr. 1983, pp. 24-41. Also *Proc. of CS292R*.
9. A. J. Thadhani, "Interactive User Productivity," *IBM Systems J.*, Vol. 20, No. 4, 1981, pp. 407-421.
10. K. McCall, "The Smalltalk-80 Benchmarks," *Smalltalk: Bits of History, Words of Advice*, ed. Glenn Krasner, Addison-Wesley, Reading, MA, 1983, pp. 151-173.
11. A. D. Samples, D. M. Ungar, and P. N. Hilfinger, "SOAR: Smalltalk Without Bytecodes,"

Proc. ACM Conf. Object-Oriented Programming Systems, Languages, and Applications, Portland, OR, Sept. 1986.

12. L. Peter Deutsch, *The Dorado Smalltalk-80 Implementation: Hardware Architecture's Impact on Software Architecture*, Addison-Wesley, Sept. 1983, pp. 113-126.
13. K. A. Pier, "A Retrospective on the Dorado, A High-Performance Personal Computer," *Proc. 10th An. Symp. Computer Architecture*, Stockholm, Sweden, June 1983, pp. 252-269.
14. Peter K. Lee, "The Design of a Debugger for SOAR," Master's thesis, Computer Science Div., Dept. of EECS, Univ. of California, Berkeley, CA, Sept. 1984.
15. L. Peter Deutsch and Allan M. Schiffman, "Efficient Implementation of the Smalltalk-80 System," *Proc. 11th An. ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages*, Salt Lake City, Utah, Jan. 1984.
16. David Ungar, "Generation Scavenging: A Non-Disruptive High Performance Storage Reclamation Algorithm," *Proc. ACM SIGSOFT/SIGPLAN SE Symp. Practical Software Development Environments*, Pittsburgh, PA, Apr. 1984, pp. 157-167.
17. Patrick J. Caudill and Allen Wirfs-Brock, "A Third Generation Smalltalk-80 Implementation," *Proc. ACM Conf. Object-Oriented Programming Systems, Languages, and Applications*, Portland, OR, Sept. 1986.



David Ungar is an assistant professor in the Computer Systems Laboratory at Stanford University. His research focuses on implementation techniques and language issues relating to object-oriented exploratory programming environments and reduced-instruction-set multiprocessors.

Ungar received a PhD in computer science from the University of California, Berkeley in May 1986. Prior to his doctoral studies, he worked in the fields of operating systems and computer networks at Bell Laboratories, in New Jersey. He also holds an MS in electrical engineering and computer science, a BS in electrical engineering, and a BS in applied mathematics and computer science from Washington University, St. Louis.



David A. Patterson is a professor in the computer science division of the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He teaches computer architecture at the graduate and undergraduate levels, and in 1982 received the Distinguished Teaching Award from the Berkeley Division of the Academic Senate of the University of California.

Patterson received a PhD in computer science from the University of California, Los Angeles in 1976. His projects have included the design and implementation of RISC 1, SOAR, and his current project, SPUR (Symbolic Processing Using RISCs. See *Computer*, Vol. 19, No. 11, Nov. 1986, pp. 8-22). SPUR investigates extending RISCs for Lisp, floating-point arithmetic, and multiprocessing. His research interests involve combining popular software, experimental architecture, and VLSI circuits to create more efficient computer systems.

Readers may write to Patterson at the Computer Science Div., 573 Evans Hall, UC Berkeley, Berkeley, CA 94720.

N . A . T . O

ADVANCED STUDY INSTITUTE

— on —

TESTING AND DIAGNOSIS OF VLSI AND ULSI

Centro A. Volta - Como (Italy) June 22-July 4, 1987

The Advanced Study Institute will cover topics on testing and diagnosis of complex devices and systems as encountered in today's high-density integration techniques. The program is aimed at up-to-date results and the latest developments with a good balance between theory and experimental design.

Demonstrations of some advanced tools are foreseen.

The lecturers, among others, will include:

J. Abraham (UI)	G. Musgrave (Brunel)
B. Courtois (IMAG)	S. M. Reddy (Iowa)
E. McCluskey (Stanford)	A. Sangiovanni Vincentelli (UCB)
M. Malek (UTA)	G. Saucier (IMAG)
J. Mucha (UH)	T. Williams (IBM)

The NATO ASI is organized and directed by:

F. Lombardi	M. Sami
University of Colorado	Dip. to di Elettronica
at Boulder	Politecnico di Milano
Campus Box 425	P.zza Leonardo da Vinci 32
Boulder, CO-80309, USA	I-20133 Milano, Italy
Tel. 1-303-492-1437	Tel. 39-2-23993516



The conference will be held at the beautiful neo-classic Villa Olmo in Como, Italy.

The NATO ASI will take place at Centro Alessandro Volta in Como (90 miles north of Milan); the facility is well equipped with computer center and lecture-hall facilities, and residential accommodations are available within walking distance.

For further information, please contact the directors

Initial Performance of the DADO2 Prototype

Salvatore J. Stolfo
Columbia University

A 1023-processor prototype named DADO2 has successfully demonstrated performance improvements for several important problems and has led to a number of avenues of future research.

On December 5, 1985 a 1023-processor parallel machine named DADO2 was successfully demonstrated at Columbia University. DADO2 is the fourth prototype, but the first large-scale prototype, of a class of machines called DADO.

DADO was first proposed in 1980 as a special-purpose parallel computer attached to a conventional host processor and designed to accelerate a particular class of artificial intelligence rule-based programming paradigms called *production systems*. PSs have been widely used for the implementation of knowledge-based expert systems.

Since its introduction, we have learned that DADO suits the high-speed execution of a large class of problems that we call almost decomposable searching problems, of which PSs are but one example. Several specific examples of this class of problems presently run on DADO2; others are currently under development and nearing completion.

In this article I present the initial measured performance of the implemented systems on the DADO2 prototype. This data is preceded by a general overview of the DADO architecture and the theoretical speedup achievable for almost decomposable searching problems.

The DADO architecture

DADO is a medium-grain parallel machine in which memory and processing are extensively intermingled. A full-scale version would comprise a large number (on the order of thousands) of *processing elements*, each containing its own processor; a reasonably small amount of local memory (20K-bytes in the current

DADO2 prototype) and a specialized I/O circuit for high-speed communication. The PEs are interconnected in a *complete binary tree*.

Within the DADO machine, each PE is capable of executing in either of two modes under the control of run-time software. In the first, called *SIMD* (for single instruction stream, multiple data stream), the PE executes instructions broadcast by some ancestor PE within the tree. In the second, referred to as *MIMD* (for multiple instruction stream, multiple data stream), each PE executes instructions stored in its own local RAM, independently of the other PEs. A single conventional host processor adjacent to the root of the DADO tree controls the operation of the entire ensemble of PEs.

When a DADO PE enters the MIMD mode, its logical state changes in such a way as to effectively disconnect it and its descendants from all higher-level PEs in the tree. In particular, a PE in the MIMD mode does not receive any instructions that might be placed on the tree-structured communication bus by one of its ancestors. Such a PE may, however, broadcast instructions to be executed by its own descendants, provided all of these descendants are in the SIMD mode. The DADO machine can thus be configured such that an arbitrary internal node in the tree acts as the root of a tree-structured SIMD device in which all PEs execute a single instruction (on different data) at a given point in time.

This discussion oversimplifies DADO2's true operation. DADO2 provides parallel remote procedure invocation in the style of SIMD processing. The procedures are stored locally within the PEs, operate autonomously, and, therefore, may take different amounts of time to complete.

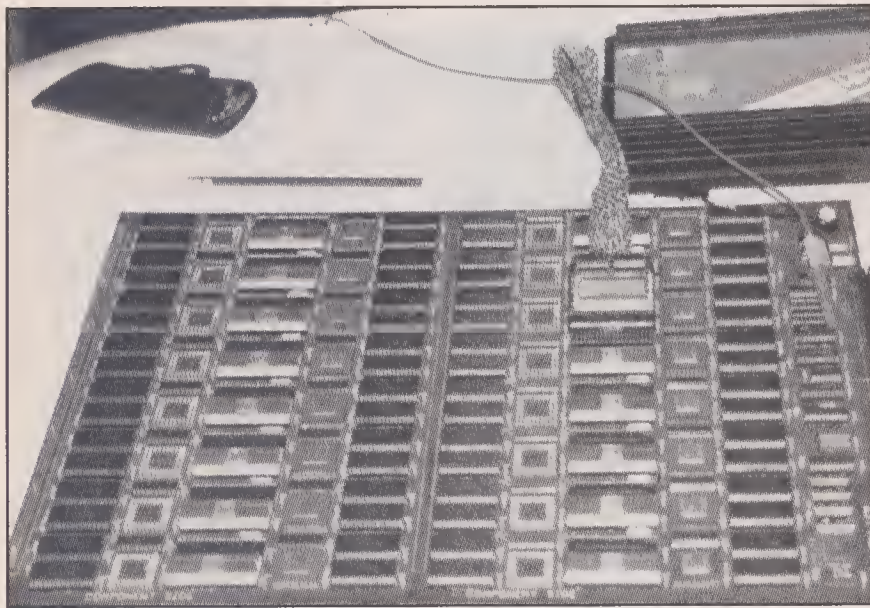


Figure 1. DADO2 circuit board with 32 processing elements.

Machine-level instructions are not broadcast and executed in lock-step. Rather, addresses of prestored code are broadcast to PEs for local execution. This mode of operation may be regarded as SPMD (for single program, multiple data stream) execution.

This flexible architectural design supports *multiple-SIMD* execution (MSIMD), whereby the machine may be logically divided into distinct partitions, each executing a distinct task.

The DADO2 I/O switch, implemented in semicustom gate array technology, has been designed to support rapid global communication. In addition, a specialized combinational circuit incorporated within the I/O switch allows for the very rapid selection of a single select PE from a set of candidate PEs in the tree, a process we call *min-resolving*. I will detail the importance and utility of these operations later.

I caution the reader not to confuse DADO with DADO2. DADO refers to a class of machines sharing certain architectural principles: tree-structured topology for fast broadcast of data, fast resolution, and a software architecture providing SIMD, MIMD, and MSIMD (or SPMD) modes of operation. The distinction between members of this family is *granularity*, the storage capacity and processor functionality at an individual PE.

DADO2 refers to a single (expedient) hardware realization of the architecture (see Figure 1) designed to experiment with parallel execution of AI/PS programs. The

suffix 2 in DADO2 refers not to the second invented architecture, but the second hardware realization (or PE design) succeeding the 15-processor DADO1. Two three-PE machines, named DADO1 3/4 and DADO1 7/8, were constructed to test various nuances of the PE circuit and were used to finalize DADO2's PE design.

The DADO/DSP, proposed by Gorin,¹ has granularity matched to large vocabulary speech recognition tasks. Raw performance data, reported later, is provided specifically for the DADO2 1023-PE prototype machine. Future physical realizations, whose engineering details will undoubtedly be influenced by experimentation with DADO2, can be expected to provide much greater performance.

DADO2 has demonstrated two important principles to date: a 1023-processor machine is relatively inexpensive and easy to build, and it works. Indeed, DADO2 was constructed within a university laboratory environment with relatively modest support.

Decomposable searching problems

Decomposable searching problems, first identified as an important class by John Bentley,² are well suited to direct execution on DADO. Indeed, Bentley later proposed, in conjunction with H. T. Kung,³ a specialized binary tree machine designed to support this class at high exe-

cution speeds. That binary tree machine comprises a number of functional units that operate synchronously upon data flowing to them, passing results to descendant PEs below in pipeline fashion.

The proposed machine does indeed provide a dual binary tree, each tree with its own root but both trees sharing the same leaves. Hence, data flows down one tree, is replicated and operated upon. Final results are then combined as data continues to flow up the second tree (see Figure 2).

DADO was independently proposed to support certain representative members of this class of problems with major architectural differences, detailed below. Much of this material is based directly upon Bentley's first paper.

A *static searching problem* is defined as follows:

- Preprocess a set F of N objects into an internal data structure D .
- Answer queries about the set F by analyzing the structure D .

Bentley defines three functions of N that characterize the complexity of the searching problem:

$S_D(N)$ = the amount of storage required by D to store the N objects of set F .

$P_D(N)$ = the preprocessing time required to build D .

$Q_D(N)$ = the time required to answer a query by searching D .

The *membership problem* illustrates this kind of problem. Preprocess N elements of a totally ordered set F such that queries of the form "is x in F ?" can be answered quickly. The common solution for serial computers is to store F in a sorted array D and binary search. Thus, the membership problem can be computed on sequential computers with complexity $S_D(N) = N$, $P_D(N) = O(N \log N)$, and $Q_D(N) = O(\log N)$.

A *decomposable searching problem* is a searching problem in which a query asking the relationship of a new object x to a set of objects F can be written as:

$$\text{Query}(x, F) = B q(x, f) \\ f \text{ in } F$$

where B is the repeated application of a commutative, associative, binary operator b that has an identity and q is a primitive query applied between the new object x and each element f of F . Hence, membership is a decomposable searching problem when cast as

$$\text{Member}(x, F) = \text{OR equal}(x, f) \\ f \text{ in } F$$

The *nearest neighbor problem*, which determines for an arbitrary point x in the

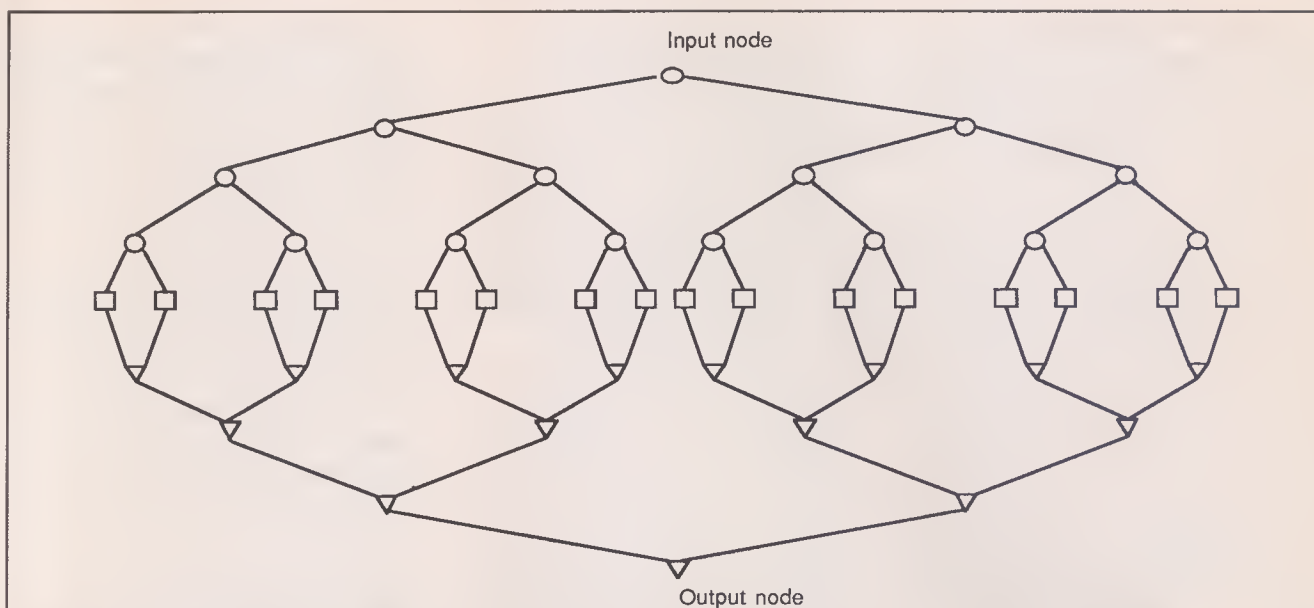


Figure 2. Bentley and Kung's tree machine.³ (Source: "Two Papers on a Tree-Structured Parallel Computer," tech. report, Dept. of Computer Science, Carnegie-Mellon University. Courtesy of the author, J. L. Bentley.)

plane its nearest neighbor in a set F of N points, can be cast as

$$NN(x, F) = \min_{f \in F} \text{distance}(x, f).$$

Based on the work of Dobkin and Lipton,⁴ Bentley points out that the nearest neighbor has serial complexity $S_D(N) = O(N)$, $P_D(N) = O(N \log N)$, and $Q_D(N) = O(\log N)$.

The key idea about these kinds of problems is *decomposability*. To answer a query about F , we can combine the answers of the query applied to arbitrary subsets of F . This characteristic also guarantees quick execution in a parallel environment. The idea is simply to partition the set F into a number of subsets equal to the number of available PEs. (For pedagogical reasons, we assume a single set element f is stored at a PE.) Apply the query q in parallel at each PE between the unknown x that is communicated to all the PEs and the locally-stored set element f . Finally, combine the answers in parallel by $\log N$ repetitions of b . This last step proceeds by applying $N/2$ b -computations simultaneously between adjacent pairs of PEs. The $N/2$ resultant values are processed again in the same fashion. Hence $N/4$ b -computations are applied in parallel, producing $N/8$ results. After $\log N$ steps the final single result is computed.

The *minimal* savings in a parallel environment are thus realized by distributing the set elements sequentially, obviating the need to sort them during preprocessing.

The query processing time, however, seems not to improve in this scheme, requiring $O(\log N)$ steps in both cases. Hence, considering the membership problem in a parallel environment under this scheme, a single query produces

$S_D(N) = N$, one element stored in each of N PEs.

$P_D(N) = O(N)$, sequential distribution as opposed to sequential sorting.

$Q_D(N) = O(\log N)$, since $\log N$ applications of b are still required.

Parallelism appears to provide modest savings only in the set-up time. This, in fact, is not the case. Significant performance savings can be realized in the following way.

First observe, as Bentley and Kung did, that a series of queries $\text{Query}(x_i, F)$, for $i = 1, \dots, M$ can be processed in pipeline fashion. In Bentley and Kung's proposed tree machine, queries flow down the tree from root to leaves and results flow up from leaves to root while being combined by operator b . Thus, after distributing F to all PEs, a single query can enter the parallel machine and begin processing immediately, followed by another query as the first query descends one level. After $O(\log N)$ steps, results begin to flow out of the parallel machine, one per time step. Thus, a series of queries would spend $O(\log N)$ time filling the pipeline, but then be computed in constant time followed by $O(\log N)$ steps to flush the pipe at the end of the series. If the number of queries is

sufficiently large, the pipe filling and flushing times can be considered negligible, and the individual query processing time constant. Hence, for large numbers of queries $S_D(N) = N$, $P_D(N) = O(N)$, and $Q_D(N) = O(1)$ for large M .

Almost decomposable searching problems

The DADO machine is quite different in its approach to solving decomposable searching problems as well as variations that we call *almost* decomposable searching problems. In a nutshell, queries are broadcast to all PEs in the tree in a constant number of instruction cycles. Primitive queries are executed in parallel by all PEs. In several important cases the combined result of applying operator b is obtained in one instruction cycle up and out of the tree. This mode of operation, referred to as broadcast/match/resolve/report, will be described with examples shortly. First, let me identify several variations of these kinds of problems to clarify the benefits of the approach taken in DADO:

(1) Consider decomposable searching problems in which the number of external queries is small, or a series of queries are generated internally from PEs within the tree. Internally-generated queries would need to migrate to the root in $\log N$ steps under Bentley and Kung's scheme, and be

broadcast down once again in $\log N$ steps. Since each query would force pipe flushing, $Q_D = O(\log N)$ for all queries. AI production systems in the style of OPS5 provide an illustration of these kinds of problems.

(2) Consider searching problems where the objects are complex or multidimensional, so that a single data structure D cannot be constructed. For example, consider relational databases where each element of a set is a complex record structure with possibly numerous fields or keys. To search such records, a data structure D would necessarily be needed for each field. Hence, in this case $P_D(N) \geq kN$ for k fields in each record.

(3) Consider a set F of first-order predicate logic literals: a knowledge base. We wish to compute a series of unifications of a series of goal literals against each of the elements of F . Since logic variables can bind to arbitrary first-order terms during unification, a single query can change the entire set of elements in the knowledge base by binding and substituting new values for variable terms. Successive queries, therefore, are processed against succeeding different sets of literals. (The simpler case involves frequent modifications to a dynamic set F , necessitating frequent reconstruction of D . Relational databases provide a convenient example.) Hence,

$$\text{Query}(x_i, F_i) = B q(x_i, f_i) \\ f_i \text{ in } F_i$$

where $F_i = \text{function}(F_{i-1}, \text{Query}(x_{i-1}, F_{i-1}))$. (In the case of logic programming, *function* is *substitution* after logical unification, while for relational databases *function* may be *insert* or *delete*.)

(4) Consider problems where a single query in a series of queries cannot be computed without knowing the result of the previous query. In dynamic programming approaches to statistical pattern-matching tasks, a single match of an unknown against the set of reference templates cannot be computed without knowing the best match(es) of the previous unknown(s). Hence, for a series of unknown x_i , $i = 1, \dots, M$,

$$\text{Query}(x_i, F) = B q(x_i, \text{Query}(x_{i-1}, F), f) \\ f \text{ in } F$$

In this case, the same pipe-flushing phenomenon appears as in the first case noted above.

(5) Finally, consider searching problems where we wish to compute a number of different queries about the same unknown x over a set, or possibly different sets.

Hence,

$$\text{Query}_i(x, F) = B q_i(x, f) \text{ for} \\ i = 1, \dots, M \quad f \text{ in } F$$

AI PSs provide an illustration of this type of problem as well.

These are the kinds of problems directly addressed and supported by the DADO machine. Parenthetically, we have, by example, identified the very set of applications for which the DADO2 machine is presently being programmed.

How do we achieve high execution speed of these tasks on DADO?

To begin with, the communication model provided by the DADO architecture assumes that broadcast of data to all PEs in the machine takes a constant number of instruction cycles. That is, communicating a single quantity (word) of data takes $O(\log N)$ electronic gate delays in DADO and not $O(\log N)$ instruction cycles via pipelined communication.

The I/O circuit of DADO2 realizes this architectural principle. A single byte of data in DADO2 is communicated from the root to all 1023 PEs in the machine in one processor instruction cycle via the I/O chip. A small constant number of additional machine instructions is necessary to move the data to its target location in memory. For data larger than one byte, a number of broadcast operations are executed equal to the length of the datum, each byte in a constant number of instruction cycles. Indeed, the I/O circuit has been conservatively designed to support one instruction cycle broadcast per byte for up to 8191 PEs.

Conversely, the I/O circuit also supports the reporting of a single byte from one PE to the root to work in a small constant number of instruction cycles. The actual elapsed time for broadcast and report operations in DADO2 is a constant 25 microseconds including the memory move operations. This key architectural principle allows for the rapid communication of data in and out of the machine. Hence, communicating a single data item from one PE to all others is achieved in constant time, not $O(\log N)$ time, and each of the above noted problems may be efficiently run on DADO.

Internally generated queries, as in case 1, can be reported and broadcast in a constant number of instruction cycles. Similarly, case 4 is efficiently handled by the same constant instruction-cycle-time broadcast/report cycle. In case 2, multiple data structures are not needed for DADO. Queries about different fields within a

record can be implemented by broadcast of the field location prior to the execution of the query. In case 3, the succeeding different sets are computed on the fly in SIMD or SPMD mode and left intact in each PE while successive queries are continuously broadcast, each in a constant number of instruction cycles.

Another key capability of the DADO machine is to provide direct hardware support for quickly computing a range of commutative and associative binary operators b . In the membership problem defined above, the binary operator OR is repeatedly applied to all of the results of the primitive query equal (x, f) . In a sequential environment, this operation may require linear time to compute. In a parallel environment, it can be computed in log time. On DADO, it is computed in constant instruction cycle time.

The I/O circuit of DADO2 provides a high speed function that we call *min-resolve*. The min-resolve circuitry calculates in one instruction cycle the minimum value of a set of values distributed one to a PE. Not only is the minimum value reported in a single instruction cycle, but the PE with the minimum value is set to what is called the "winner state," indicating the entire ensemble of loser PEs as well as the single winner PE in the computation. (Ties are arbitrated in hardware according to a fixed PE ordering scheme.)

The membership problem can thus be solved by applying min-resolve to zeros and ones (distributed throughout the tree after complementing the result of the equality operator) to compute OR. The nearest neighbor can be computed by applying min-resolve to the distances. (Similarly, data generated within DADO can be read in sorted order in $O(N)$ time by sequential enumeration of min-resolved winners.)

Finally, note that problems characterized by the fifth example can be efficiently supported by the MSIMD mode of operation. In this case, DADO can be arbitrarily partitioned into a number of smaller sub-DADOs, each executing a distinct task. Each sub-DADO can thus process its own query, while a forest of sub-DADOs can compute a number of queries concurrently.

Let us now review how the DADO machine can quickly execute decomposable searching problems as well as the more difficult class identified by the five observations. Membership will provide the vehicle for illustration.

(1) Preprocess N objects of set F by distributing each element in turn to one PE of DADO. Note, $S_D(N) = N$ for the entire machine and $P_D(N) = N$.

(2) Broadcast the unknown object x to all PEs in a constant number of instruction cycles.

(3) Apply the query equal (x, f) in parallel at each PE (in SIMD or SPMD mode). In parallel, each PE sets the min-resolve value to 0 if $x = f$, otherwise it sets it to 1.

(4) Min-resolve in one instruction cycle.

The overall computation time for $Q_D(N) = O(1)$, the sum of steps 2, 3, and 4. In cases where the primitive query q is more complex than equality, the running time is proportional to the slowest of the entire set of concurrent executions of q .

Production systems on DADO. Another illustration is provided by AI PSs. In general, a PS is defined by a set of rules, or *productions*, which form the *production memory* (PM), together with a database of assertions, called the *working memory* (WM). Each production consists of a conjunction of condition elements, called the *left-hand side* (LHS) of the rule, along with a set of actions called the *right-hand side* (RHS). The RHS specifies information to be added to (asserted) or removed from WM when the LHS successfully matches against the contents of WM.

In operation, the PS repeatedly executes the following cycle of operations:

(1) Match: For each rule, determine whether the LHS matches the current environment of WM. All matching instances of the rules are collected in the *conflict set of rules*.

(2) Select: Choose exactly one of the matching rules according to some predefined criterion.

(3) Act: Add to or delete from WM all assertions specified in the RHS of the selected rule.

An AI PS can be cast as an almost decomposable searching problem by repeatedly applying the query

$$\text{Query}(\text{WM}_i, \text{PM}) = \text{MAX match}(r, r \text{ in PM})$$

WM_i) for $i = 1, \dots, M$ for some M

where $\text{WM}_i = \text{WM}$ on the i -th cycle computed by applying the selected rule on the $i-1$ st cycle to WM_{i-1} . (WM_0 is the initial WM.) Match is a function that calculates whether a rule matches the current WM and rates the quality of the match by some predefined criteria.

One of the simplest reported algorithms (named the Full Distribution Algorithm)

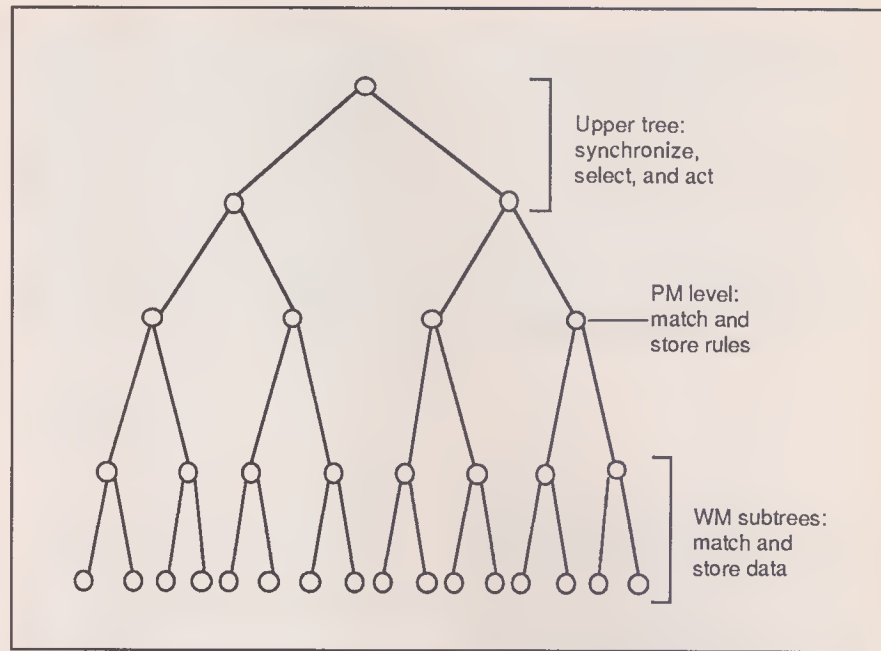


Figure 3. Functional division of the DADO tree. (Source: "Initial Performance of the DADO2 Prototype," *J. of Parallel and Distributed Computing*, Vol. 3, No. 2, June 1986, p. 279. Courtesy of the publisher, Academic Press.)

for PS execution on DADO works by distributing a single rule to each of the available PEs. Data in WM is broadcast in turn to all PEs, which match it against the stored rule, saving those data elements that match some condition element of the LHS of their rule. Rules whose LHSs are thus fully matched are then resolved to determine the single rule instance to execute. The RHS actions of the selected rule are then broadcast to all other PEs, initiating another match cycle. In effect, the entire rule base is fully distributed into a collection of one-rule PSs processed in parallel.

In certain contexts, the primitive query match can itself be cast as an almost decomposable searching problem. The match of a single rule requires the match of each condition element in its LHS against the database of WM elements. Thus, given the stylized rule

$$C_1 C_2 \dots C_M \rightarrow A_1 \dots A_n,$$

match can be considered a *conjunction of queries*:

$$\text{Query}_j(C_j, \text{WM}_i) = \text{OR primitive-w in WM}_i$$

$$\text{match}(C_j, w), \text{ for } j = 1, \dots, M$$

or, when variable bindings between condition elements are possible,

$$\text{Query}_j(C_j, \text{WM}_i) = \text{OR primitive-w in WM}_i$$

$$\text{match}(C_j, \text{Query}_{j-1}(C_{j-1}, \text{WM}_i), w) \text{ for } j = 2, \dots, M.$$

Primitive-match is the lowest level test of an individual condition against a specific data element in WM. This leads to yet another parallel algorithm for AI PSs utilizing the MSIMD mode of operation of DADO. In this case, each rule matched appears in the root PE of a sub-DADO. These root PEs appear at one level of the machine called the PM-level (see Figure 3). Each sub-DADO, and hence each distinct rule, is matched in parallel against the WM elements fully distributed to the WM-subtrees below. The final accumulated result of the concurrent rule matchings is computed by a higher-level parallel process similar to the one depicted in the first PS example.

The stylized algorithm (named the Original DADO Algorithm) proceeds as follows:

- (1) Initialize: Distribute a match routine and a distinct rule to each PM-level PE. Set Changes to the initial WM elements.
- (2) Repeat the following:
- (3) Act: For each WM-change in Changes, do
 - (a) Broadcast WM-change to the WM-subtree PEs that modify themselves accordingly.
 - (b) Initiate the parallel match phase in each PM-level PE:
 - (i) Each condition element of the rules (the C_j noted above) stored at a PM-level PE is

broadcast to the WM-subtree below for primitive matching.

- (ii) A local conflict set of rules is formed along with a priority rating.
- (c) End do.
- (4) Select: The min-resolve circuit is used to identify the maximally-rated conflict set instance from all PM-level PEs.
- (5) Report the instantiated RHS of the winning instance to the root of DADO.
- (6) Set Changes to the reported action specifications.
- (7) End Repeat.

These algorithms, as well as other variants, have been reported elsewhere⁵ and are either presently running or under development on DADO2.

The question to ask at this point is, precisely what speedup can be achieved for AI PS programs? Much has been written and debated about this very point. The issue centers upon two key observations.

First, the match function itself is executed in parallel by a large number of PEs. In certain measured OPS5 PS programs, it has been reported that a relatively small and stable number of rules must be matched on each cycle of PS execution. That is, on average each rule firing in OPS5 produces new data elements that affect (or are relevant to) the LHS of a small number of rules. (Statistics for R1, for example, show that about 30 rules out of a set of approximately 2000 are affected on each cycle of execution.) This small affect set, as it has been called, thus indicates that the number of individual match operations executing in parallel that compute useful new matching rule instances is small. Hence, a large number of invoked parallel procedures compute nothing at all and the utilization of the parallel resources is poor. (The well-known RETE match algorithm for OPS exploits this observation by utilizing a discrimination network data structure—D in the characterization above—that links rules to be matched with newly inserted WM elements. Sequential OPS is thus executed much more efficiently than purely naive sequential matching approaches.)

The second observation to note is that the variance in processing times for the concurrent match operations can be quite large. That is, the total running time is proportional to the slowest match operation, indicating that utilization of the parallel resources may decrease even further while possibly many PEs wait for the slower PEs

to catch up and synchronize for the resolve operation.

These two observations may lead one to the depressing view that PS programs, in particular OPS5, can best be accelerated by a small number of parallel processors rather than the large-scale approach taken in the DADO machine. We on the DADO project believe this very point requires further investigation. The DADO project at Columbia has discovered and is implementing two clever source-to-source transformational techniques as well as a multitasking approach that provide the means to reduce the rule-matching variance time as well as increase the average affect set size.

Unfortunately, space does not permit a detailed review of this work. I encourage the reader to refer to other papers^{6,7} for details of these approaches. The essence is to rewrite the rule system in such a way that individual rules that require more processing than the average rule are replicated a number of times with additional distinct constraints appearing in each copy. This transformation preserves the same computation as the original rule, but provides the opportunity to match the copied rules in parallel. The total effect is that a number of rules representing the single "hot spot" rule can more quickly calculate the same result in parallel. Results reported in the last section show how one system was optimized in this fashion to produce a speedup of 31 over a VAX computer.

The second technique to increase the average affect set size depends upon concurrent rule firings, where the select phase chooses for parallel execution as large a number of rule instances as possible, rather than a single rule instance. A multitasking approach to this parallel activity on DADO2 has been reported⁸ with encouraging simulation results noted: a factor of 7.5 times faster execution for a 237-rule OPS5 program.

Another technique is quite similar to the notion of "chunking" introduced by Rosenbloom.⁹ Whereas chunking has been proposed as an AI learning method, the same technique may be used to coalesce a number of rules into larger rules with larger RHSs. The net effect is to rewrite a set of individual rules often executed serially to a new larger rule (encompassing all the others) whose actions compute the same WM changes and hence create larger affect sets in one cycle of execution. Consequently, the execution of a single chunked rule, in place of its fore-

bears, can lead to a larger number of useful match operations calculated in parallel.

One experiment performed to date provides evidence to support the utility of the third approach. An OPS5 program, named Puzzle2 (appearing later in Table 2), ran nearly twice as fast on DADO2 using the chunking approach. Combined with the copying and constraining technique, Puzzle2 ran 18 times faster on DADO2 than on a VAX. Hence, it is definitely possible to enhance performance through the application of program transformation techniques to extract further parallelism than appears on the surface. Work is presently under way to apply these techniques to a larger OPS5 program to be run on DADO2.

One final word about PS performance. Much of the debate that has arisen is based upon statistics derived from the performance of existing sequential OPS5 programs.¹⁰ Many parallel processing researchers concur that opportunities for exploiting parallel processing may not be derived from the study of only sequential programs. After all, sequential programs were written for sequential processors, not parallel processors. This may lead researchers toward techniques of conventional program optimization to improve code efficiency on a single PE, the primary approach taken in the implementation of OPS83.¹¹ We prefer to optimize for increased parallel performance. Hence, a potentially more useful approach may be to consider rewriting existing sequential code entirely with the mind's eye set on parallel processing. (Fortunately, not many lines of OPS code exist in comparison to Fortran, for example.) To that end, expression of parallelism is necessary and hence parallel languages are vitally important.

Much of the current activity of the DADO project at Columbia focuses on the specification and implementation of an AI PS language, called HerbAI, designed to be downward compatible with OPS5. The essence of the initial HerbAI definition provides a few additional parallel constructs to permit the expression of inherently parallel activities that can be expressed only sequentially in OPS. The expected net result is to provide improved expression as well as improved performance of AI programs (see van Biema¹² for details of some initial experiments). We expect that the initial HerbAI effort will provide a brighter future for large-scale parallel processors.

Programming DADO

The other examples identified above (logic programming, relational database operations, and statistical pattern recognition) can be similarly defined as almost decomposable searching problems. Each of these problems shares the same common programming paradigm on DADO:

- Distribution of an initial set of data.
- Broadcast of data in a constant number of instruction cycles.
- Local execution of some primitive query against a prestored element of the reference set (unification of first-order logic literals, relational selection of tuples, and dynamic time-warp match of a template of a speech utterance, respectively).
- Resolution to find the best answer in a constant number of instruction cycles.
- Report of the final answer.

To date, three high-level parallel programming languages are available on DADO2 for expressing these computations. Each is based upon the same principle, whereby a small set of parallel processing and communication primitives are added to a familiar high-level conventional language. Hence, parallel PL/M (PPL/M), parallel C, and parallel PSL Lisp (PPSL) all share a common expressive formalism for parallelism, but a different base language. These languages have been reported elsewhere.^{13,14} I demonstrate the Member example in parallel C to convey the flavor of the current style of programming DADO2. (This example forms the basis of the relational selection implementation whose performance I detail later.)

SIMD variables (f in the following example) correspond to vectors whose components are fully distributed to the PEs, that may be operated upon, in whole or in part, in parallel.

```
simd char f;  
  
member(x)  
char x;  
{  
    return (minresolve(broadcast(x) != f));  
}
```

In the following section I detail the performance of two almost decomposable searching problems appearing in relational database operations and OPS5 PSs. Statistical pattern recognition examples will be reported elsewhere. Logic programming examples are not yet available.

Table 1. Timings in seconds for SELECT * FROM <table> WHERE <field> = <literal> on DADO2 and IBM 4381. The key is a fixed length 5 character string. Timings were averaged over 10 runs for each size reported.

Number of Records	IBM 4381 SQL	DADO2	Speed-up
1K	0.042	0.0056	7
4K	0.141	0.0224	6
8K	0.265	0.0448	6

Performance

The contenders in this tournament are

- an IBM 4381 running VM/CMS in a main memory of 16M bytes,
- a VAX 750 running BSD Unix 4.2 in a configuration of 6M bytes of main memory,
- the DADO2 prototype machine with 1023 8-bit processing elements, each with 16K bytes of user memory, totaling roughly 16M bytes. DADO2 was hosted in all cases by a VAX 750 running Unix 4.2.

The relational database benchmarks were run on the IBM mainframe using IBM's SQL system. The AI PS benchmarks were run on the VAX alone using a compiled version of CMU's widely distributed OPS5 system. All benchmarks were run on DADO2 using parallel C with a VAX 750 host running Unix. Each benchmark is described in turn.

Relational database operations. The primary operation tested was relational selection on a variety of different sets of 32-character tuples. Each relation ranged in size from 1023 to 8191 tuples in the benchmarks reported here. All times reported in Table 1 are absolute times for precisely the same operations running on each machine.

Disk transfer rates are intentionally ignored in these timings. Since DADO is designed as an attached peripheral, we intend the machine to simply offload processing from a mainframe's processing unit. Hence, external-to-internal transfer rates are considered the same for a mainframe both with and without the DADO peripheral. I therefore report only the performance of the two machines executing purely searching code for the relational selection task against databases contained solely in main memory.

As Table 1 indicates, DADO2 is six times faster in selecting data over the IBM

4381 processing unit. Clearly, selection runs fast as a purely decomposable searching problem. Why is DADO2 not 1023 times faster than the IBM 4381?

One key observation must be made and understood before proceeding. The theoretical speedups discussed earlier must be viewed in relation to a single PE within DADO2. That is, 1023 PEs should be 1023 times faster than one PE. Each DADO2 PE is a simple 8-bit Intel 8751 processor pitted against a huge Goliath IBM 4381. Even so, 1023 Davids beat the Goliath six times over for selection.

Rating the single PE's performance against the IBM would provide perhaps more useful data. However, executing the programs reported here in a single DADO2 PE would be quite difficult if not impossible due to the severely limited address space available for the prototype PE. Hence, we measure DADO2 in comparison to conventional computers with sufficient capacity to store the same programs in their main memories. The following benchmarks should be understood in this context. (Our next stage of research will be devoted to measuring speedups directly in terms of processor utilization to provide a better understanding of the speedups achievable by parallel processors such as DADO.)

AI production systems. We measured the time to process RHS actions of rules, additions and deletions of WM elements, and the time to match the newly formed database against the rule base. The TREAT algorithm, developed by Miranker,¹⁵ was implemented in the style of the first PS algorithm discussed above (Full Distribution). No optimizations were performed on the DADO2 code. The rules were loaded sequentially and randomly distributed throughout the machine. Thus, the rated performance must be viewed as an upper bound and will undoubtedly significantly improve in the

Table 2. DADO2 versus a VAX 750 for OPS5 rule matching (timings in milliseconds).

PS Name	#Rules	#WM	#DADO PEs	VAX	DADO2	Speed-up over 750
Mapper	237	528	287	68,119	9,696	7
Monkeys	13	12	13	2,240	754	3
Puzzle 1	13	65	137	313,344	28,663	11
Puzzle 2	13	65	137	313,344	17,850	18
Puzzle 3	13	65	438	313,344	10,028	31
Waltz	33	50	50	77,350	34,873	2
Mud	884	241	943	675,172	338,844	2

future when our optimization techniques are fully implemented.

In several of the cases reported below, the copy-and-constrain technique was used with encouraging results. As noted, one program implemented utilized the chunking technique and produced significant performance gains, providing evidence of the efficacy of this approach as well.

In the present implementation, only the match phase (which processes the LHS of distributed rules) is executed directly on DADO2. The instantiation of the RHS of the selected rule is performed by the host VAX. This initial implementation pays a modest price in performance. The instantiation of the selected rule is reported to the VAX in its entirety. The VAX then proceeds to compute the instantiation of the RHS (stored in its memory). The cost of communicating the entire instantiated LHS could be avoided by having the winning PE in the selection phase perform this computation directly. This bottleneck has become our primary target for further performance gains in the current implementation effort.

The programs executed consisted of from 13 to 884 OPS5 rules and from 12 to 528 WM elements. Measurements were made of the total time to execute the rule system over a large number of rule execution cycles.

Included in Table 2 are the number of PEs used in the computation on DADO2. This indicates the number of additional copied and constrained rules introduced to the initial systems to boost performance. Notice that in no case have the fully available DADO2 PEs been used in these initial benchmark programs. Hence, only a fraction of the machine has been rated.

Of particular interest is the performance of the programs called Mapper and Puzzle3, respectively 7 and 31 times faster

than a VAX 750. These programs perform best on DADO2 simply because of the amount of matching tasks they require that can be executed in parallel. A perusal of Table 2 confirms that the larger the number of data elements to match relative to the number of rules, the more matching tasks that need to be executed. Hence, the speedup is greater, in general, for programs that have proportionately more data elements than rules. Note, for example, that Mud is only twice as fast on DADO2 as on the VAX, where 241 data elements are matched against over 800 rules.

The performance of Puzzle2 confirms that the approach of chunking rules may provide greater performance advantages. When combined with a smaller degree of copying and constraining in the Puzzle1 program, a multiplicative effect was measured where overall performance was boosted to a factor of 18, up from a factor of 11.

It is clear from these initial performance figures that indeed there are additional sources of parallelism and concomitant speedup to be explored for AI PSs. Earlier pessimistic statistics published for OPS5 must thus be considered too negative.

Our next year of research is devoted to producing a wide range of experiments on a large number of OPS5 programs to fully understand the effects of these various optimization techniques. We are encouraged to believe by these initial measurements that even greater performance gains for DADO2 are thus achievable.

As a final note: DADO2 has initially displayed a modest range of performance gains over two serial machines for two almost decomposable searching problems. There is one very important question that should not be lost by any parallel architect: At what cost? Here

is where DADO2 is perhaps at its best. DADO2's physical characteristics and hardware complexity are roughly equivalent to a VAX 750 (including the age of their respective technologies). DADO2, however, is much smaller and less expensive than an IBM 4381. Indeed, DADO2 is considered an attached peripheral for each machine. In the case of the VAX, hardware is doubled (although the full DADO2 machine is not employed) for at least a factor of from 2 to 31 gained in performance for OPS5 rule-matching for a modest collection of programs. In the case of an IBM 4381, a small add-on peripheral produces a factor of six gained in performance for relational selection.

This data suggests that DADO is a good cost/performance peripheral for almost decomposable searching problems. Data to be reported elsewhere by researchers at AT&T Bell Laboratories showing significant speedup of statistical pattern recognition tasks on DADO2 will substantiate this claim as well. □

Acknowledgments

I gratefully acknowledge the contributions of Dan Miranker, Rick Reed, Russell Mills, Philip Yuen, Mark Lerner, Mike van Biema, and Lee Woodbury to the research reported here. I also thank Pandora Setian for her editorial assistance.

This work has been supported by the Defense Advanced Research Projects Agency under contract NOO039-84-C-0165, and the New York State Science and Technology Foundation under contract NYSSTFCAT(84)-15, as well as by grants from DEC, Hewlett Packard, IBM, AT&T, Intel, and Valid Logic Systems.

References

1. A. L. Gorin, J. E. Shoenfelt, and R. N. Lewine, "Speech Recognition on the DADO/DSP Multiprocessor," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, Tokyo, Japan, Apr. 1986.
2. J. L. Bentley, "Decomposable Searching Problems," *Information Processing Letters*, Vol. 8, No. 5, June 1978, pp. 244-250.
3. J. L. Bentley and H. T. Kung, "Two Papers on a Tree-Structured Parallel Computer," tech. report, Dept. Computer Science, Carnegie-Mellon Univ., Sept. 1979.
4. D. Dobkin and R. J. Lipton, "Multidimensional Searching Problems," *SIAM J. Computing*, No. 5, 1976, pp. 181-186.
5. S. J. Stolfo, "Five Parallel Algorithms for Production System Execution on the DADO Machine," *Proc. Nat'l Conf. AI, AAAI*, Univ. Texas at Austin, Aug. 6-10, 1984, pp. 300-307.
6. S. J. Stolfo, D. P. Miranker, and R. C. Mills, "More Rules May Mean Faster Parallel Execution," *Proc. Workshop on AI and Distributed Problem Solving*, Nat'l Academy of Sciences, Washington, DC, May 16-17, 1985, pp. 101-108.

7. T. Ishida and S. J. Stolfo, "Towards the Parallel Execution of Rules in Production System Programs," *Proc. 1985 Int'l Conf. Parallel Processing*, IEEE, Aug. 20-23, 1985, pp. 568-575.
8. T. Ishida and S. J. Stolfo, "Simultaneous Firing of Production Rules on Tree Structured Machines," tech. report, Dept. Computer Science, Columbia Univ., Mar. 1984.
9. J. E. Laird, P. S. Rosenbloom, and A. Newell, "Towards Chunking as a General Learning Mechanism," *Proc. Nat'l Conf. AI, AAAI*, Univ. Texas at Austin, Aug. 6-10, 1984, pp. 188-192.
10. A. Gupta and C. L. Forgy, "Measurements on Production Systems," tech. report, Dept. Computer Science, Carnegie-Mellon Univ., 1984.
11. C. J. Forgy, "OPS83 User's Manual and Report," tech. report, Production Systems Technology, 1985.
12. M. van Biema, D. P. Miranker, and S. J. Stolfo, "The Do-loop Considered Harmful in Production System Programming," *Proc. First Int'l Conf. Expert Database Systems*, Institute of Information Management, Technology and Policy, Univ. South Carolina, Columbia, SC, Apr. 1-4, 1986.
13. S. J. Stolfo, D. P. Miranker, and M. D. Lerner, "PPL/M: The System Level Language for Programming the DADO Machine," tech. report, Dept. Computer Science, Columbia Univ., Feb. 1984.
14. M. van Biema et al., "PSL: A Parallel Lisp for the DADO Machine," tech. report, Dept. Computer Science, Columbia Univ., Feb. 1984.
15. D. P. Miranker, "Performance Estimates for the DADO Machine: A Comparison of TREAT and RETE," *Proc. Int'l Conf. Fifth Generation Computer Systems*, Tokyo, Nov. 1984, pp. 449-457.



Salvatore J. Stolfo is an associate professor of computer science at Columbia University. His main fields of interest are artificial intelligence, knowledge-based expert systems, and parallel architectures. Stolfo received an IBM Faculty Development Award for his research work. Working in close collaboration with researchers at AT&T Bell Laboratories, he codesigned and implemented a large-scale expert system marketed by AT&T Technologies, Inc.

Stolfo received a BS in computational mathematics from Brooklyn College in 1974 and MS and PhD degrees in computer science from Courant Institute, New York University, in 1976 and 1979, respectively.

Readers may write to the author at the Computer Science Dept., Columbia University, New York, NY 10027.

PROGRAMMING LANGUAGE EXPERTS

Tomorrow's Computing Technology is Today's Challenge

at IDA

Some of the nation's most exciting developments in software technology, supercomputer architecture, AI, and expert systems are under scrutiny right now at the Institute for Defense Analyses. IDA is a Federally Funded Research and Development Center serving the Office of Secretary of Defense, the Joint Chiefs of Staff, Defense Agencies, and other Federal sponsors.

IDA's Computer and Software Engineering Division (CSED) is seeking professional staff members with an in-depth theoretical and practical background in Programming Languages. Tasks include efforts on both compiler-based tools for program development and testing and on language validation. Tasks include efforts both in prototyping tools and in advising the DoD on policies for language development and maintenance. The emphases of the Division's efforts are in Ada and logic programming languages.

Specific desired interests and skills include theory and practical experience in:

- Compilers and compiler generators
- Code optimization
- Language validation
- Formal semantic representation
- Formal representation for requirements/designs

- Design and rationale of Ada
- Language directed dynamic computer architectures

Specialists in other areas of Computer Science are also sought: **Software Engineers, Distributed Systems, Artificial Intelligence and Expert Systems, and Computer Security Specialists.**

We offer career opportunities at many levels of experience. You may be a highly experienced individual able to lead IDA projects and programs . . . or a recent MS/PhD graduate. You can expect a competitive salary, excellent benefits, and a superior professional environment. Equally important, you can expect a role on the leading edge of the state of the art in computing. If this kind of future appeals to you, we urge you to investigate a career with IDA. Please forward your resume to:

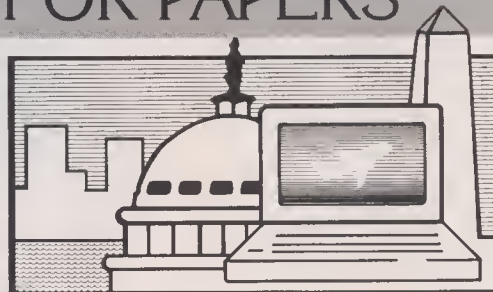
Mr. Thomas J. Shirhall
Manager of Professional Staffing
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

An equal opportunity employer.
U.S. Citizenship is required.



ADVANCE CALL FOR PAPERS

THE THIRD ANNUAL EXPERT SYSTEMS IN GOVERNMENT CONFERENCE



TENTATIVE DATE: October 19-23, 1987
WHERE: Metropolitan Washington, DC
THEME: Are expert systems living up to their promise?

Sponsored by the Mitre Corporation, the Computer Society of the IEEE, and the George Washington University.
In Cooperation with AIAA National Capital Section.

TOPICS OF INTEREST

The conference focuses on the use and development of expert systems by government agencies. The topics of interest include, but are not limited to, the following AI techniques and application areas:

- Knowledge Representation
- Knowledge Acquisition
- Reasoning Under Uncertainty
- Expert Systems for Robotics
- Image Understanding
- Natural Language and Intelligent Interfaces
- Expert System Development Environments
- Distributed and Parallel Architectures
- Command and Control Applications
- Logistics Support Systems
- Financial and Project Management
- Nuclear Industry Applications
- Aerospace Applications
- Maintenance and Diagnostic Systems
- Software Lifecycle Applications
- Weather Forecasting

FOR FURTHER INFORMATION, CONTACT:

Conference Chair:

R. Peter Bonasso, The Mitre Corporation, bonasso@mitre.arpa

Conference Vice-chair:

Jude Franklin, PRC Government Information Systems

Technical Program Co-chairs:

Papers:

Jim Antonisse, The Mitre Corporation, jima@mitre.arpa (703) 883-7887

Jack Benoit, The Mitre Corporation, benoit@mitre.arpa (703) 883-6511

Panels:

Barry G. Silverman, Institute for AI, George Washington University (202) 676-6433

Tutorials Co-chairs:

Steven W. Oxman, OXKO Corporation

Lisa Sokol, The Mitre Corporation, sokol@mitre.arpa

Address of the conference chairs is:

ESIG '87

MS W418

The Mitre Corporation

7525 Colshire Dr.

McLean, VA 22102

Computer Society of the IEEE

1730 Massachusetts Avenue, N.W.
Washington, DC 20036-1903

SUBMISSION INFORMATION

Submit three copies of the paper by May 15, 1987 to

ESIG '87

MS W418

The Mitre Corporation

7525 Colshire Dr.

McLean, VA 22102

The first page of the paper should contain the author's name, affiliation, address, electronic address if available, phone number, and 100 word abstract. Accepted papers will be allocated up to six manuscript pages, approximately 4800 words, in the proceedings. Authors will be notified of acceptance or rejection by July 1. Camera-ready copy for accepted papers is due by August 1.

Proposals for panels and tutorials related to the topics of interest are also solicited.

Applications of the Connection Machine

David L. Waltz

Thinking Machines Corporation

This massively parallel system is easily programmable, applicable to a variety of applications, and can be scaled to large problems by the addition of more processors.

The Connection Machine development effort¹ was initiated in the belief that parallel processing and artificial intelligence could together accelerate the rate of progress toward truly intelligent machines. In the brief time since the first Connection Machine was switched on in April 1985, we have begun to realize this dream at a pace and with an ease that has far exceeded our expectations. This progress is the result of the ease with which the machine can be programmed and the dramatic increase in compute power that the machine can bring to bear.* We have been able to run many trials of experiments in instances where previously, running just one would have been considered an achievement and no further experimentation would have been done. This has enabled us to explore a great many more hypotheses and to work on much larger problems than had been possible on previous-generation artificial intelligence workstations. The ease of programming is in part the result of a decision to use existing serial machines (the Symbolics 3600 or Digital Equipment Corporation VAX), thus leaving unchanged the operating systems, editors, file systems, debuggers, network communications systems, and so on, so as to provide familiar programming environments. The Connection Machine is programmed in conservative extensions of Common Lisp and C. Users familiar with these languages and with front-end computer systems have been able to produce results on the Connection Machine on the first day that they use it.

*The Connection Machine can perform most operations at a rate in excess of 1000 MIPS; 32-bit addition operates at more than 2000 MIPS. As a comparison, note that a DEC VAX 11/780 operates at about 1 MIPS. (MIPS = millions of instructions per second.)

Data-level parallelism. Programming the CM has proven to be easy. Because there are so many processors, it is practically impossible to program each one separately. Instead, the programming method we have generally used, called *data-level parallelism*, is similar to that used on ordinary serial machines.** In this method, each element of data for the problem is stored, and one element is stored per processor (or virtual processor). The front-end computer executes a serial program, each step of which can involve computations in all of the Connection Machine's processors. For example, rather than using loops for repetitively computing operations on an array or set of data elements, one serial program carries out the operations simultaneously in all of the processors.

Such a solution seems far more natural and easy to construct than those typically proposed or implemented for coarse-grained parallel machines. Coarse-grained parallel computers (ones with two to a hundred or so processors) must generally be programmed by means of a method that might aptly be called "control-level parallelism." Control-level parallelism requires a programmer to divide a program into fragments, one for each of the processors in the coarse-grained machine. Synchronization is a problem in coarse-grained systems; moreover, it is often not easy to find parallelism in programs, let alone exactly the right amount of parallelism—that is, an amount that can be distributed equally among the number of available processors.

**Not all Connection Machine programs exploit data-level parallelism. For an example of Connection Machine programming that assigns different operations to different processors, see Blleloch.²

In a wide variety of cases, we have demonstrated that the Connection Machine can exploit data-level parallelism to achieve high computation rates. Data-level parallelism is particularly appropriate for database, natural-language, and image-processing operations, for finite-element modeling, graphics, and automata-based modeling of physical processes.

Hardware and software: system-level specifications. The CM system is at the "radically fine-grained" end of the spectrum of parallel machines; it is available in configurations ranging from 16K (2^{14}) processor-memory units up to a maximum of 64K (2^{16}). The processor-memory units are interconnected with a high-speed communications network that allows all processors to communicate with any or all other processors that are in parallel. The 64K Connection Machine system consists

of the processor-memory-communications system plus a front-end machine.

A Connection Machine system supports up to eight front-end computers. Currently, each front end is either a Digital Equipment VAX or a Symbolics 3600 Series Lisp Machine. Physically, the Connection Machine consists of 4096 chips of proprietary design, each containing 16 processors plus a hardware router. The CMs on which most of the benchmarks in this article were run consist of 16K processors, each with 8M bytes of memory (4K bits per processor). Processors operate in a bit serial fashion: each operation can combine two bits from memory with one bit from a register, producing one bit to memory and one bit to a register. Each of these basic load-load-store operation cycles requires about one microsecond.

The processors are organized as a 12-dimensional hypercube with 16 processors at each vertex of the hypercube. A

packet-switching network handles communication between the processors. Each 16-processor chip also contains a network router. Each router can accept a message from one of the processors on its chip (or from a router on a different chip) and send it to either a processor on its chip or another router on a different chip. Each router can also buffer messages if there are no channels available over which to send them. A message header consists of 16 bits; each bit corresponds to a dimension along which the message must move in order to be delivered properly. The header can be computed by comparing the 16-bit address of the source processor with the 16-bit address of the destination processor, putting ones in each dimension in which the addresses differ. Each time the message is routed along a dimension of the machine, the bit corresponding to that dimension is set to zero. When all the bits are zero, the message has arrived at its

Connection Machine Document Retrieval System

Top of TEXT

****PM-PRESIDENT-PROPERTY****
MARCOS PROPERTY ON SALE FOR \$300 MILLION, NEW YORK TIMES SAYS
NEW YORK, Feb 22, Reuter - Four Manhattan buildings believed owned by Philippine President Ferdinand Marcos and his wife Imelda have been on the market since last June for up to \$300 million, the New York Times reported today.
All are owned by corporations situated in countries outside the United States which keep secret the identity of the owners, although Congressional testimony has indicated they are controlled by the Marcoses, the newspaper said.
It said documents from the Security Pacific National Bank indicated the owners "wanted to sell the properties because of financial difficulties, management disputes, and sensitivity to political criticism in Philippines about holdings in the United States."
Although the documents do not reveal the potential profit from the sale of the buildings, testimony before the House Foreign Affairs Subcommittee on Asian and Pacific Affairs showed that Mrs. Marcos told one building manager in 1984 she hoped to make \$70 million by 1987, the report said.
The Marcoses' wealth was a significant issue in the Feb. 7 presidential elections, although the couple has denied owning property in the United States. Some congressmen have said they want to restrict further aid to the Philippines until the Marcoses repatriate their American investments.
According to the report, discussions on the sale of the properties began about the time that articles began to appear about their holdings last June.
NEC reported Thursday that the Marcoses have been trying for the past nine months to sell off huge portions of their art and U.S. real-estate holdings.
Signs that the couple were trying to convert their assets into cash first popped up last spring when major art works began appearing on the market, NEC said.
Among the paintings for sale was a Monet, which had been bought by Mrs. Marcos in London in the 1970s, offered for between \$2 million and \$2.5 million, the network said.
Cash from the sale was deposited into a Swiss bank account, it said.

Bottom of TEXT

Top of DOCUMENTS

1. NEWS PICTURES ADVISORY
Good Bad [2297 Reuters 1/03/86]
2. AQUINO DENOUNCES MARCOS IN PRESIDENTS OWN B
Good Bad [2834 Reuters 1/02/85 BAGUIO, Philippines]
3. MARCOS TRIES TO LAUGH OFF U.S. ARMY CHARGES
Good Bad [5117 Reuters 1/24/85 MANILA]
4. PHILIPPINES ASSAILS REPORT ON FAKE MARCOS WAR
Good Bad [6120 Reuters 1/23/85 MANILA]
5. PHILIPPINES ASSAILS REPORT ON FAKE MARCOS WAR
Good Bad [6181 Reuters 1/23/85 MANILA]
6. AQUINO DEATH THREAT REPORTED AS RAIN WRECK
Good Bad [6315 Reuters 1/21/85 MARAWI, Philippines]
7. **MARCOS PROPERTY ON SALE FOR \$300 MILLION,**
Good Bad [5594 Reuters 2/22/85 NEW YORK]
8. U.S. ECONOMIC INDEX MAKES SLIM NOVEMBER GAIN
Good Bad [632 Reuters 12/30/85 WASHINGTON]

Bottom of DOCUMENTS

Top	Bottom	Up	Down
REUTERS NEW-DEMO-8-8 Waiting for Command			
Help	Options	Flush	Refresh
Other	Seed Words	Search	Exit

HPLUS, WEALTH

Top
Bottom
Up
Down

10/28/88 15:36:41 DB-DEMO
USER:
101

Figure 1. The search process on the Connection Machine document-retrieval system. Here, the user had typed "Marcos" and "Wealth." The first eight documents containing "Marcos," "Wealth," or both are shown in the upper right window. The user has marked article number seven as good by clicking on its mouse-selectable "Good" box.

destination. Messages can be combined by means of four operations: *or*, *max*, *add*, or *overwrite*. Typical message delivery times when all processors send messages are on the order of 250 microseconds for 32-bit messages; times range from 70 microseconds for very regular communications patterns to a worst case of 900 microseconds when all processors send to other processors at random.

Parallel channels support I/O rates of up to 2000M bps. The channel supports very-high-speed devices, such as disks and frame buffers.

Each processor can, under software control, mimic a number of virtual processors. To accomplish this, the memory of each processor is divided up so that each "virtual processor" operates on a smaller amount of memory. We have routinely programmed applications involving a million virtual processors. The speed penalty for computation is approximately linear. Thus, going to n virtual processors per processor slows down the computation by a factor of n ; there is very little overhead.

The CM can be programmed at the front end with C*, which is an extension of C, or *LISP, an extension of Common Lisp. To the programmer, the front-end machine appears essentially unchanged: The same file systems, editors, debuggers, network protocols, and so on are available. The only difference is that a set of additional instructions and parallel structures is available in each programming language. There are parallel analogs of most common instructions. For example, the expression

(+ !!a b)

can simultaneously add and store: It can carry out the addition of variables stored in field a in each of 64K processors to the contents of each of 64K instances of the variable b , and store the result on an internal stack within each processor. Other *Lisp constructs are also natural extensions of well-known operations. For instance, (*if *test operations*) executes the *test* and then executes the *operations* only in those processors in which the test succeeded. (The "currently selected set" of processors may, of course, be further restricted by including another "*if" among the operations.) Examples of *Lisp and C* code for the Connection Machine are given in the section on "Programming a document-retrieval system," below.

The raw computing rate of the machine is on the order of 64,000 MIPS for single-

bit instructions. If the machine is operating on units larger than a bit (for instance, a 32-bit number) then the raw computation speed must be divided by the number of bits in the word to achieve a measure of the actual computing rate. Thus, the CM performs 32-bit adds at a rate of approximately $64,000 \div 32 + 2000$ MIPS. For database applications, the relevant word length may be only one byte, and may sometimes be as short as a bit. As a result, for these applications we have benchmarked very high performance, approximately equivalent to a serial machine operating at 6000 MIPS.

Document retrieval

We have built an easy-to-use document-retrieval system that mixes AI ideas with methods derived from the information-science research of the 1960's.³ Its basis is a weighted associative-memory algorithm. Document retrieval has traditionally been implemented as Boolean search on an inverted file. The main difficulties of Boolean search are

- users require considerable training in the use of a query language, and
- users generally alternate between being overwhelmed by too many documents (if one uses a search pattern that is too general) or being given too few documents (if one's search pattern is more restrictive).

Boolean search systems also exhibit poor precision and recall.⁴ In contrast, a naive user can be trained to use our system in a few minutes; the system operates very rapidly; and the system has high precision and recall.

Document retrieval by "relevance feedback." From the user's point of view, the search process on the Connection Machine document retrieval system has two distinct phases.

In the first phase, the user types a list of a few keywords or a question. In Figure 1, for example, the user has typed "Marcos," and "Wealth." The front-end machine broadcasts coded forms of each of these two words to all processors; it also broadcasts numerical weights. Each processor notes whether each word occurs in its article; if a word appears, it increments the score of its article by the appropriate numerical score. The weight assigned to a word is inversely proportional to the log of its frequency in the database. (For example, "platinum" appears in the database

less frequently than "gold," and therefore has a higher associated weight). This weighting mechanism ensures that uncommon words have more of an influence than common words do over which content kernels get selected during the document-lookup process. As long as the terms in the search profile are all from closely related articles, the longer the search profile, the more precise the search.

When all the words have been broadcast, the articles with the largest scores are retrieved and presented to the user.

In the second phase, the user can browse through the documents returned and find one or more documents that bear on the topic of interest. Our interface allows the user to use a mouse to mark documents as "good" or "bad." Once one or more documents have been marked, the user can repeat the search again. This time, *all* the words in the document or documents selected are broadcast. The documents that are most relevant will share a large number of words with the selected, or "seed," document. The results of such a search are shown in Figure 2. This method, termed *relevance search*, has been known since the 1960's to have excellent properties of precision and recall, but to the best of our knowledge, it has never been used in a practical system because of its extremely high computational requirements.

The algorithm takes advantage of the innate characteristics of documents. Every document is, in effect, a thesaurus for its subject matter. A high percentage of the synonyms for each topic appear because writers work to avoid repetition. In addition, variants of words (such as plural, singular, and possessive forms), and semantically related terms also appear among the words in a particular article. Clearly not every synonym, variant, and related term will occur in a single article, but many of the related terms will. The use of many terms to form a search pattern makes the search highly selective. While the word "chip" may appear in an article about cookies, "VLSI," "integrated," and "circuit" simply will not. In the overall scoring, truly useful documents are reliably separated from random matches.

Seed-document search requires only about 40 milliseconds on the Connection Machine, given a 200-term search pattern and about 3K bytes of free text per CM processor. A more detailed discussion of performance appears in the section on "Timing and performance," below.

Building a database on the Connection Machine system. The document database is constructed from sources of text, such as wire services, electronic mail, and other electronic databases. The system we have used for benchmarking contains about 15,000 Reuters newswire articles. One article is stored per processor; each article averages on the order of 3000 bytes of ASCII data. The text of the articles is compressed by means of techniques described below, so that the 3000 bytes are encoded in about 500 bytes. For that description, it is important to draw a distinction between *source documents* and *content kernels*. A source document contains the full actual text of a particular article, book, letter, or report, and it is stored on the file system of the front-end machine. A content kernel is a compressed form that encodes just the important words and phrases in a source document. It omits the commonplace words. Content kernels are stored in the

memory of the Connection Machine system.

The content kernel is produced automatically from the source document. First, the source document is processed by ConText, a Thinking Machines Corporation document-indexer program that selects the most significant *terms* (phrases) in the text. The resulting list of terms contains, on the average, about 130 words per article—a data compression of about 3.5:1 over the original article. Next, these words are encoded into a bit-vector data structure by means of a method called *surrogate coding*. In surrogate coding, each word in the content kernel is mapped into n different bits in the 1024-bit vector as a result of the use of n hash functions. If more than one word hashes into a particular bit of the bit string, the result is simply *or-ed*. For our database, $n=10$. For 35 terms, the process of surrogate coding ends up marking about a third of

the bits as ones, adding another factor of 2:1 to the data compression.

Surrogate coding allows the content kernel to be stored more compactly. It also speeds up the search process. To test for the presence of a given word within a document, the same 10 hash functions are applied to the word, generating 10 bit positions. The 10 bit positions are then checked for each of the approximately four bit vectors representing each article, and if any of them match, the score of the overall article is incremented. This scheme is probabilistic; it is possible that all 10 bits for a word might have been set because of other words, even if that word doesn't really appear in the source document. Such an accident will result in a "false hit" on that word. However, this will not seriously affect the result of the lookup, for two reasons. First, the probability of a false hit is small: about $(\frac{1}{2})^{10}$, or less than one in 50,000. Second, a false hit will be

Connection Machine Document Retrieval System

Top of TEXT

****AM-MARCOS****
MARCOS TRYING TO SELL ART, U.S. REAL ESTATE, NEC SAYS
NEW YORK, Feb 20, Reuter - Philippine President Ferdinand Marcos and his wife, Imelda, have been trying for the past nine months to sell off huge portions of their art and U.S. real estate holdings, NEC Nightly News reported today.
Signs that Marcos and his wife were trying to convert their assets into cash first popped up last spring when major art works began appearing on the market, NEC said.
Among the paintings for sale was a Monet, which had been bought by Mrs. Marcos in London in the 1970s, and was offered for \$2 million to \$2.5 million, the network said.
Cash from the sale was deposited into a Swiss bank account, it said.
The network, citing confidential sources who included art dealers, said pieces of an Asian art collection owned by Mrs. Marcos were also coming up for sale.
Many galleries and auction houses refused to handle the art objects because of the secrecy surrounding them and questions on the legitimacy of their title, NEC said.
Also put up for sale were some of the U.S. real estate holdings Marcos and his wife own, it said. The total value of their U.S. real estate empire has been estimated in Congress at \$350 million.
Marcos, who has ruled the Philippines for some 20 years, was recently declared the winner in a bitterly fought election. But his opponent, Corason Aquino, insists she won and is calling on him to step down.

Bottom of TEXT

Top of DOCUMENTS

1. **MARCOS PROPERTY ON SALE FOR \$300 MILLION,**
Good Bad [8594 (Reuters 2/22/85 NEW YORK)]
2. **MARCOS TRYING TO SELL ART, U.S. REAL ESTATE, N**
Good Bad [2744 (Reuters 2/20/85 NEW YORK)]
3. **SOLARE SAYS MARCOS SHOULD BE FORCED TO SELL**
Good Bad [7640 (Reuters 2/03/85 WASHINGTON)]
4. **SOLARE SAYS MARCOS SHOULD BE FORCED TO SELL**
Good Bad [7643 (Reuters 2/03/85 WASHINGTON)]
5. **IMELDA MARCOS SAID TO SEEK \$70 MILLION FROM IN**
Good Bad [6599 Reuters 1/29/85 WASHINGTON]
6. **LAWYER, BANKER SAY IMELDA MARCOS OWNS NEW**
Good Bad [6192 Reuters 1/23/85 WASHINGTON]
7. **MARCOS MAY OWN MILLIONS IN U.S. PROPERTY, PANI**
Good Bad [4155 Reuters 1/16/85 WASHINGTON]
8. **MARCOS OWNS MILLIONS IN U.S. PROPERTY, CONGREI**
Good Bad [4140 Reuters 1/16/85 WASHINGTON]

Bottom of DOCUMENTS

Top	Bottom	Up	Down
REUTERS NEW-DEMO-8-8 Waiting for Command			
Help Other	Options Seed Words	Flush Search	Refresh Exit
SEED DOCUMENT SEARCH			

Left: Select Document, Right: menu of Edit Document, Print Document, Save Document, Analyze Match, Cluster, Select Document.
10/28/86 15:38:52 DB-DEMO USER: Toy

Figure 2. Results of seed-document search. Shown at the right are the top eight documents—those most similar to the document selected in Figure 1. (Note that since only one document was used to form the search pattern, it appears first because it is perfectly similar to itself.) Scores for the documents are shown to the right of the "Good" and "Bad" boxes. Any number of "Good" and "Bad" selections can be batched together. Terms in "Bad" documents that do not appear in "Good" ones are given negative weights.

only one of many terms contributing to the score, and so will have only a small effect even when it does occur.

The source document in its original form is available for retrieval by and presentation to the user when needed. The location of the original document on the system disk is stored with the content kernel.

Programming the document retrieval system

The Connection Machine system's easy programmability is probably its greatest advantage over its competition. The figures referred to in this section illustrate this point by showing program code for the document retrieval system described above. Both *Lisp and C* versions of the program are included. Programs for several of the other applications in this article can be found in *Introduction to Data-Level Parallelism*.⁵

The content kernel's data structure is made up of the following fields:

- **score** is used by the document lookup program to accumulate the ranking of each content kernel in the database. Rank is awarded according to how closely the content kernel matches the user's search pattern. Each time a match is found, **score** is updated.

- **document-id** contains a reference to the original source document from which a content kernel was derived. When a content kernel is selected, the user is shown the source document referred to by this field.

- **kernel** is a table of the surrogate-coded bit-vector encoding.

The necessary declarations for these fields are shown in Figure 3. (All of the code for this example is presented twice, first in the *Lisp language and then in the C* language, to make it easy to compare the two languages. Because the characters "*" and "?" may not appear in C* identifiers, such *Lisp names as **score** and *word_appears?* are rendered in C* simply as *score* and *word_appears*.)

The modifier *poly* declares variables present in all processors. There is also a modifier *mono*, which declares variables resident in the front-end machine.

Document lookup on the Connection Machine system. During the first stage of document lookup, the user lists a set of

terms to be used to search the database, and receives back a list of documents that contain all or some of those terms. The user then points to a document that is relevant, and from this document an overall *search profile* of content-bearing words is assembled. The search profile is simply a list of these words, with weights assigned to each word. Next, the search profile is broadcast to all processors in the Connection Machine system. For each word in the search profile, a set of 10 bit indices (*bit-locs*) is broadcast. All content kernels that have these same 10 bits set have the weight of that word (*word-weight*) added into their **score** field.

The code shown in Figure 4 accomplishes this.

The main search program simply calls this routine once for each keyword in the keyword list. Since *word_appears* is a parallel variable, its handled in the following way when it follows an *if*: each processor whose *word_appears* value satisfies the test is marked as belonging to the currently selected set of processors by the pushing of a stack bit in each processor. Operations broadcast by the front-end machine are always carried out only by processors in the currently selected set. When one exits the scope of the conditional, the stack is popped, and the previous currently selected set is restored.

```
;;; Declarations for the *Lisp version.

(defconstant table-size 1024)
(defconstant hash-size 10)
(*defvar *score*)
(*defvar *document-id*)
(a) (*defvar *kernel*)

/* Declarations for the C* version. */

#define TABLE_SIZE 1024
#define HASH_SIZE 10

poly unsigned score, document_id;
(b) poly bit kernel [TABLE_SIZE];
```

Figure 3. Declarations necessary for the **score, **document-id**, and **kernel** fields. (a) The *Lisp version. (b) The C* version.**

Retrieving the highest scoring documents. The code in Figure 5 is used to retrieve the **document-id** for each of the highest scoring content kernels in the database. The program returns a list of **document-id**s for the content kernels with the highest scores. The program first retrieves the **document-id** for the highest score, then for the next highest score, and so on, until a list of length

```
;;; *Lisp code for testing the presence of a single word.

(*defun increment-score-if-word-appears (bit-locs word-weight)
  (*let ((word-appears? t!))
    (dolist (bit bit-locs)
      (*set word-appears?
        (and!! word-appears?
          (not!! (zerop!! (load-byte!! *kernel* (! bit) (! 1))))))
      (*if word-appears?
        (*set *score* (+!! *score* (! word-weight))))))
(a)

/* C* code for testing the presence of a single word. */

poly void increment__score__if__all__bits__set
  (mono unsigned word__bit__position [HASH_SIZE], mono int weight) {
  mono j;
  poly bit word__appears = 1;
  for (j = 0; j < HASH_SIZE; j + +)
    word__appears &= kernel [word__bit__position[ j ]];
  if (word__appears)
    score + = weight;
(b) }
```

Figure 4. Code for testing a document for the presence of a single word. (a) The *Lisp version. (b) The C* version.

document-count is retrieved. The *already-retrieved?* flag is set once a processor has had its **document-id** retrieved so that the **document-id** will not be retrieved again.

Timing and performance. Benchmark tests using Reuters newswire articles have been run for document search. The tests were performed on a 16K-processor Connection Machine. Each document was typically represented by approximately 130 content words. The content words were selected by ConText, a Thinking Machines Corporation automatic indexing system, from the roughly 400 words in an average article. A seed-document search is based on the union of the sets of content words from several articles, and typically involves between 200 and 300 words. Only 40 milliseconds was required to search 15,000 articles (representing about 40M bytes of free text) against a

200-term pattern and to assign scores to each article; in another 20 milliseconds, the system can select the 20 articles that most nearly match the set of 200 terms. Exactly the same amount of time would be required to search four times as many documents (64,000 documents, the equivalent of 160M bytes of free text)* on a 64K-processor Connection Machine.

The amount of computation that would be required to do the benchmark calculation on a serial machine is staggering: For each document, one would have to compare each word with the list of 200 terms in the search pattern, and for each term that matched, a score would have to be added to the document's overall score. On the average, on a serial machine, only about 1.488 bits of a 1024-bit surrogate code vector would have to be tested in order to find

out that a term does not occur in it, assuming that 10 hash functions are used to encode each term, and that one-third of the bits in the vector are set.

On a serial machine, one could test for the states of the hashed bits corresponding to a term one at a time, until one of the tested bits was a zero. For any document in which the term occurred, one would have to make 10 tests. If one third of the bits in each 1024-bit vector are set to one, there is a two-thirds chance that the first bit tested is zero, a two-thirds chance that the second bit is zero, and so on. Thus, the total number of tests that must be made is

(1 test <always> + ($\frac{1}{3} \times \frac{2}{3}$ <just one more test>) + ($\frac{1}{9} \times \frac{2}{3} \times 2$ <more tests>) + . . . + ($\frac{1}{3^8} \times \frac{2}{3} \times 8$ <more tests>) + ($\frac{1}{3^9} \times 9$ <more tests>) = 1.488 tests.)

Thus, assuming that each term searched for occurs in 1 percent of the documents, then, on the average, for a database consisting of 64,000 documents and for a search profile of 200 words, a serial program would have to execute the following number of instructions:

64,000 documents $\times$ 4 vectors/document $\times$ 200 search terms $\times$ [($.99 \times 1.488$ tests/search term/vector <assuming a document miss> + $.01 \times 10$ tests/search term/vector <assuming a document hit>) $\times$ 3 instructions <increment, AND, test, and jump at a minimum> $\times 1.1$ <to allow for addition of partial results, incrementing outer loops, and so on> = 265.8 million instructions.

In order to carry out this operation in 40 milliseconds, one would need to have a serial machine that operates at $(1/.04) \times 2.658 \times 10^8 = >6645$ MIPS.

Even databases that are larger will be able to use the same technique—with two enhancements. The first enhancement is the use of a very high-speed paging disk currently under development; the disk will allow larger numbers of content kernels to be swapped into the system for searching. The second enhancement is the use of *cluster analysis*. When the system has many documents on the same subject, it need not store all their content kernels individually. It can store one for the whole cluster, then retrieve the full set of related documents when these are needed. A single document may, of course, participate in more than one cluster. As the total database size grows, the size of the average cluster grows with it, making this a particularly appropriate technique for large-

*Note that 160M bytes is equivalent to an entire year of news from a typical newswire.

;;; *Lisp code for retrieving documents in order, highest score first.

```
(*defun retrieve-best-documents
  (let ((top-documents-list nil))
    (*let ((already-retrieved? nil))
      (dotimes (i document-count)
        (*when (not!! already-retrieved?)
          (*when (= !! *score* (*max *score*))
            (*let ((next-highest-document (*min (self-address!!)))
              (setq top-documents-list
                (append top-documents-list
                  (list (pref *document-id* next-highest-document))))
              (self (pref already-retrieved? next-highest-document) t))))))
    top-documents-list))
```

(a)

/* C* code for retrieving documents in order, highest score first. */

```
poly void retrieve__best__documents
  (mono document__count, mono unsigned *document__id__array) {
  poly bit already__retrieved = 0;
  mono i;
  for (i = 0; i < document__count; i + +) {
    if (!already__retrieved) {
      if (score == (>= score)) {
        processor *next_highest_document = (<= this);
        document__id__array[i] = next_highest_document->document_id;
        next_highest_document->already__retrieved = 1;
      }
    }
  }
}
```

(b)

Figure 5. Code for retrieving the **document-id** for each of the highest scoring content kernels in the database. (a) The *Lisp version. (b) The C* version.

scale databases. The addition of paging and clustering will extend the algorithm described above to the 10-gigabyte range and beyond.

Memory-based reasoning systems

One of the unrealized goals of artificial intelligence is the creation of programs that base their "reasoning" on experience. AI researchers have assumed that most of the relevant experience of human experts is "encoded" in the form of rules. While there has been some progress toward the goal of automatically generating rules of inference,⁶ the construction of intelligent programs has for the most part proven to be a difficult task, one that depends on the ability of highly skilled individuals to create sets of rules.

We believe that memories of specific events, and not rules, are the key to reasoning from experience. For example, a program could diagnose diseases by comparing a patient's symptoms and characteristics with those of all the patients it knew about; if it could "remember" similar patients with similar symptoms, those patients could be used as a basis for reasoning about the current patient. Memories of particular instances have not been used in the past as a basis for reasoning because sequential computers are too slow to execute some of the necessary basic operations, for example, finding on nontrivial amounts of data it stores the item most similar to an input item. (This capability is sometimes called *associative memory*.) It is important that memory-based reasoning systems work with large amounts of data, so that it will in general be possible to find instances in memory that are close to the current instance. With new parallel architectures, specifically with the Connection Machine system, such operations suddenly become sufficiently fast, even with large amounts of data, to use as the basis of reasoning.

The reason that this solution has been largely overlooked in the past is that von Neumann machines do not support it well. There have been systems, such as Samuel's Checkers player,⁷ that work from memory, but they have required (a) an exact match between an item in memory and the current situation and (b) a nongeneral search algorithm (for example, one for hashing or indexing) to retrieve items from memory.

In the real world there may be no exact matches, so a "best match" is called for. Unfortunately, there is no general way of searching system memory for the "best match" without examining every element of memory. On a von Neumann machine, this makes the use of large memories impractical. On a fine-grained parallel machine, such as the Connection Machine system, this problem does not arise until the size of the database exceeds the size of the machine's memory; this pushes the problem back three or four orders of magnitude.

While it is not yet possible to encode experience in the manner used by humans, it is possible to build practical systems that use databases as memory. Like the document-search system described in the previous section, a memory-based reasoning system depends on the capability of the CM to search globally for the nearest match to items in a database. The system also needs to be able to compute statistical operations (for example, counting and sorting) on large numbers of items very rapidly. An example of the CM's power in this domain: It is able to sort large numbers of items in a period of time proportional to the square of the log of the number of data items. To do this, it uses methods similar to the APL programming language's "plus-scan," which is also referred to as "parallel prefix," "sum-prefix," or "recursive doubling."⁸ Sets of up to 64,000 items can be counted or added together in about 300 microseconds. By means of related operations, 64K items can be sorted in about 30 milliseconds. This ability makes statistical computation tractable, even for large amounts of data. The resulting CM system can bypass the problems of knowledge acquisition and brittleness that have plagued expert system researchers.

The operation of memory-based reasoning. The key operation in memory-based reasoning is weighted associative search for items in memory that are similar to a present case that one hopes to understand or act appropriately on. Once some matches (or near matches) have been found, they can suggest *hypotheses*. Other operations, which are based on statistics, test these hypotheses; to do so, they separate important *features* from unimportant ones.

The operation of memory-based reasoning systems can be made clearer through an example. Suppose that we have a large relational database of medical

patient records, with fields for age, sex, symptoms, features of patient history (for example, pack-years smoked, previous heart attack(s), and so on), diagnosis, test results, treatment, and so on. I will refer to the entries in these fields as "features." A program can find diagnostic hypotheses for a new patient by

- (1) broadcasting each feature of the new patient,
- (2) having each processor compute a numerical measure of closeness to each feature, and then having the processor add this number to a total similarity score, and
- (3) selecting from the database the patients whose total scores are closest to that of the new patient.

There are several possible outcomes:

- (1) all n similar patients received the same diagnosis,
- (2) no patients are very similar to the new patient,
- (3) only a small number are similar, and
- (4) (the general case) there are several diagnoses among the nearest n patients.

In all four cases, we next run a hypothesis-testing phase. If its results suggest that only one diagnosis is plausible, we are done; if more than one diagnosis remains, a second hypothesis-testing method is invoked, which either completes the diagnosis, or proposes tests that would differentiate among the remaining possibilities. (These processing phases are described in more detail below.) However, even at this point, we can make some observations:

- If (1) is true (that is, if all n patients have the same diagnosis), it is very likely that this patient should also get the same diagnosis, and it is likely that we could even extract a rule for diagnosing this disease that could be added to a microcomputer-based expert system.

- If case (2) holds, the system knows that it may not have ever seen this set of symptoms before, and thus, it "knows that it doesn't know" how to diagnose this case. (Alternatively, the data on the new patient may be in error, and the system can also propose this possibility.)

- In case (3), even if only one patient is similar (for example, as in the second reported case of Legionnaire's disease), the system may be able to make a reasonable hypothetical diagnosis. Approaches taken in the event of case (4) are described below.

Table 1. Four cases from the Quack database.

Field	1	2	3	4
Eye-Color	Brown	Gray	Brown	Brown
Hair-Color	Blond	Brown	Black	Brown
Sex	Male	Female	Female	Male
Aches	-	-	-	-
Backache	-	-	-	-
Chills	-	-	-	-
Nausea	-	-	-	-
Period-Late	N/A	1	2	N/A
Running-Nose	-	-	-	-
Sneezing	+	-	-	-
Sore-Throat	+	-	-	-
Tired	+	-	-	-
Fever	Mild	None	None	None
P-Test	-	-	+	-
T-Culture	-	-	-	-
U-Test	-	-	-	-
Condition	Throat	Healthy	Pregnant	Healthy

A system for medical reasoning. The idea of memory-based reasoning has been successfully demonstrated in a preliminary test of feasibility.⁹ A medical reasoning system called Quack was constructed. Since the creators lacked quick access to a real database of medical cases, one was created: a stochastic model that draws on the creators' commonsense knowledge of common medical conditions (hence the name!). This example is not offered as compelling evidence of the correctness of our theories, only as an illustration and as a limited demonstration of their feasibility.

Quack simulates the process of diagnosing a patient. It starts with a database of past cases, including symptoms, personal data, results of tests, and the ultimate diagnosis. It is then given information about a series of patients, starting with personal data and symptoms. First, it forms and evaluates a set of hypotheses, eventually narrowing them down to a set of between one and three possibilities. It then selects some tests (for example, taking the patient's temperature), and uses the test results to make a final determination. Quack does all this without being given any rules or other information; it has only the database of past cases.

The database is created by a multiphase stochastic process. First, personal attributes (such as sex, hair, and eye color) are chosen at random. Sex is a significant feature, while hair and eye color are added in an effort to distract the system. Second,

a medical condition is generated. The possible conditions include pregnancy, influenza, colds, over-exertion, good health, and a variety of other conditions. Third, directly observable symptoms are generated. These symptoms include generalized aches and pains, backaches, chills, nausea, and sneezing. The symptoms of various conditions overlap, symptoms are absent in some cases, and symptoms unrelated to the primary condition sometimes appear (for example, tiredness and aches). Finally, the results of several tests that a physician might order are predicted, including the results of taking a patient's temperature and taking a throat culture for strep throat. Table 1 shows four cases from this synthetic database.

The basis of Quack is the retrieval from a database of cases similar to that of the current patient. The most important aspect of this retrieval problem is the metric used in computing similarity. This metric must give low weight to irrelevant features, such as hair and eye color; high weight to definitive features, such as the results of pregnancy tests; and intermediate weight to suggestive features, such as aches and chills. Brief reflection indicates that no one metric is adequate. For example, if a patient is suspected of having the flu, his or her sex is completely irrelevant and should be given zero weight. On the other hand, in diagnosing pregnancy, sex is definitive, and should be given extremely high weight. Thus, the metric is

dependent on the system's hypothesis.

The metric is calculated "on the fly." Given a hypothesis, the system will determine how well each feature correlates with the hypothesis, and choose weights accordingly. For example, if Quack hypothesizes that a patient has the flu, it will compute the proportion of males having the flu, the proportion of females having the flu, and note that sex is not a significant predictor for influenza. Thus, it will assign a low weight to sex. On the other hand, if Quack hypothesizes that a patient is pregnant, it will note that men are never pregnant, that females are sometimes pregnant, and give sex a weight sufficiently high to exclude men from consideration.

The next step is to find the cluster of cases most similar to that of the current patient. This is done by applying the metric to every case in the database, then extracting the cases most similar to that of the patient. Once the cluster of similar cases has been retrieved, the proportion of cases matching the hypothesized diagnosis is calculated. This may strengthen or weaken the hypotheses. In addition, the composition of the cluster will be used to generate additional hypotheses, which will be evaluated in turn.

This process of hypothesis generation and evaluation is illustrated by the following sample run. The test patient was male and was experiencing a sore throat, chills, and tiredness. Quack started with the assumption that he was healthy. This hypothesis generally yields a good initial metric and, while the hypothesis of good health is usually rejected, comparison with those cases in which patients are unhealthy in a similar manner yields good hypotheses. In this case, several hypotheses did emerge, including the flu, nonspecific throat infection, and strep throat. Each of these hypotheses was tested in turn. The only ones that survived this stage of the evaluation were nonspecific throat infection and strep throat. Table 2 shows the output of this phase of the trial. It may be noted that implausible hypotheses are generated. This is not directly harmful because the hypothesis-evaluation step is sufficiently strong to weed out the bad hypotheses.

Once the initial diagnosis is made, several alternatives may remain, in which case a second phase of reasoning is necessary. First, the resolving power (definitiveness) of various tests is computed, relative to the possibilities generated in the first phase. Second, those tests that have high resolving power are selected for use in a sec-

ond phase of hypothesize-and-test. In the example we have been looking at, two tests were selected: taking a throat culture and taking the patient's temperature. When these added factors were taken into account, a solid diagnosis was made. A trace of this phase is shown in Table 3.

This example required 8.9 seconds to run on a database of 4096 examples. It was also run on a database of 32,768 examples, where it took 10.7 seconds to run and generated fewer false hypotheses. It was also run on a database of 512 examples, where it took 4.3 seconds to run but misdiagnosed the case (the small database contained only one case of strep). In general, increasing the size of the database improves the quality of the system's reasoning without significantly changing the time needed for generating hypotheses. For very small databases, the system's performance deteriorates somewhat, but the system remains capable of handling the more common problems. Recent research has added learning to the system, allowing it to perform much more accurately and rapidly. The new system precomputes weightings, and can generate hypotheses with a single lookup operation.

It is possible to criticize Quack on the ground that all its deductions are extremely obvious. But that is precisely the point: The basis of intelligent behavior *must* include a grasp of the obvious, something that is sorely lacking in rule-based systems. Remember, Quack does not have a knowledge base, only a collection of weak methods for making decisions based on memory. It also has the useful property that, as its base of experience grows, it gets smarter rather than stupider: The quality of its decisions improves without incurring a significant time penalty.

Memory-based reasoning was also tested more recently—on a word-pronunciation task very similar to NETtalk's.¹⁰ Its performance was somewhat superior to NETtalk: The memory-based reasoning system we built, called MBRTalk, was able to generate understandable pronunciations for about 65 percent of the novel words it was given; the basis of its reasoning was the known pronunciations of about 4500 randomly selected English words. This research is described by Stanfill and Waltz.⁹

Evaluation of memory-based reasoning. Computer learning has generally stayed within the heuristic search/deductive reasoning paradigm: Most algorithms for learning try to derive a set of rules from a body of data. This approach is exemplified

by the work described by Michalski, Carbonell, and Mitchell.¹¹ Memory-based reasoning, in its pure form, differs from traditional learning methods in that no rules are ever created (although, as mentioned above, it can be used to generate rules, in a manner similar to Quinlan's ID3.⁶) Reasoning from examples, rather than from rules, has several advantages. First, the deductions made by the system are achieved without the intermediate use of rules as a representation, so there are fewer opportunities for inadvertently introducing inaccuracies, for example, those that result from combining "confidence levels" for a chain of rules. Second, rule generation has a combinatorially explosive search space of possible rules to contend with, while memory-based systems can focus their analysis on the case at hand. It may be that in capturing a large knowledge base, so many rules are needed that the rule-generation process will never finish. If this is the case, there is never any certainty that the resulting rule set is complete, and the resulting system may fail at any moment. With a memory-based system, the data used as the basis of reasoning is always available. Therefore, within limits imposed by the accuracy of the retrieval mechanism and the contents of the database, errors of omission should not occur.

Space does not allow an extended discussion of this work in rule-based reasoning. It is closely related to other research—in parallel associative memory, semantic nets, and "case-based reasoning."¹²

Memory-based reasoning systems may be able to function like expert systems in domains where there are databases of situations and outcomes/actions. Possible application areas include medical diagnosis and proposal of diagnostic tests, weather forecasting, deciding whether to extend credit to a customer, investment advising, insurance-risk assessment, and many others. No expert is required; a knowledge engineer only needs to identify database contents and to mark them according to whether they are symptoms or features of situations; outcomes or actions; or optional tests that can be ordered. Memory-based systems can form hypotheses on the basis of even a single precedent, something rule-based systems cannot do—rules are inherently summaries of regularities among classes of items. And memory-based reasoning systems "know when they don't know": If no items in memory are closely related to the item being analyzed, then the system is able to recognize this fact and to tell the user.

Table 2. Initial diagnosis of a patient as performed by Quack.

Likelihood	Healthy	0.0
Suggests	Throat, Strep, Flu	
Likelihood	Flu	0.0
Suggests	Pregnant, Ov-exer, Cold, Ut-inf*	
Likelihood	Ut-inf*	0.0
Likelihood	Cold	0.0
Likelihood	Ov-exer	0.0
Likelihood	Pregnant	0.0
Likelihood	Strep	8.031373
Likelihood	Throat	33.573772
Resultant likelihood of hypotheses:		
Throat	0.8070	
Strep	0.1930	

*"Ut-inf" stands for urinary tract infection.

Table 3. Quack's diagnosis of a patient after the taking of a throat culture.

Tests	Fever, T-culture	
Likelihood	Throat	0.0
Likelihood	Strep	3.766667
Resultant likelihood of hypothesis:		
Strep	1.0000	

Bulk processing of natural language

A natural-language processing system that finds indexable phrases in natural language text has been implemented on the Connection Machine. (This system was mentioned earlier as a component of the document-retrieval system.) The system operates in two phases, as illustrated in Figure 6. In the first phase, the definition for each word is looked up. For the purposes of this task, the definitions consist of part of speech information for many words; for other words, the definition simply states that the word should be deleted because it cannot be part of an index. For example, most adverbs and words such as "if," "among," and "between" can never be part of an index term. Word definitions also mark whether the

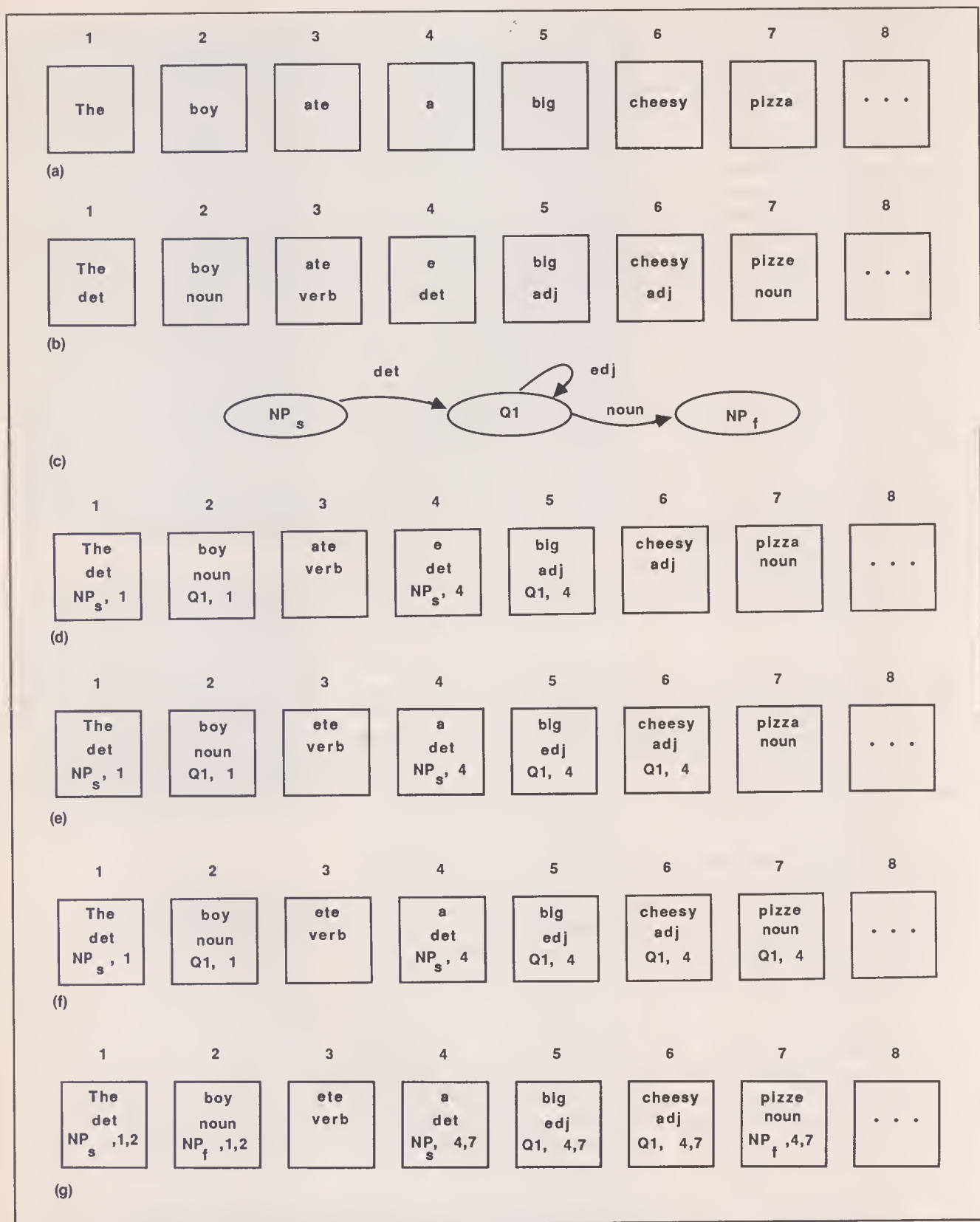


Figure 6. Parallel bulk processing of text in the CM natural-language processing system. (Parts (a), (b), and (c) constitute phase one; parts (d) through (g) constitute phase two.) (a) Original text. (b) Parts of speech assigned in parallel (see text). (c) Sample ATN grammar. (d) Determiners recognized, and Q1 symbol sent. (e) First adjective recognized. (f) Second (and last) adjective done; no adjectives are processed in the next step, so the noun arc is broadcast. (g) Noun phrase done; phrase start and end locations are stored in each cell.

word is the name of a famous person, a day of the week, the name of a month, and so on. In the second phase, a series of ATN (Augmented Transition Network) parsers is run over all the text simultaneously; during this phase, all noun phrases are found simultaneously.

Let us suppose that we have a simple ATN grammar¹³ that specifies that a noun phrase consists of a determiner followed by any number of adjectives followed by a noun. Let us assume that each word has been assigned to a virtual processor, and that parts of speech have been stored for each word. This grammar is used to parse all of the noun phrases that match it, in the following manner: First, an instruction is broadcast to all processors; it says that if the processor contains a determiner, then the address of the processor (the starting address for the noun phrase) and a special state symbol, "Q1," should be sent to the processor to the right of the given processor. (The processor to the right of the given one holds the next word of the text.)

In the next phase, all processors that contain symbol Q1 are selected and instructed to check the definition(s) of their word. If the word can be an adjective, then the symbol Q1 and the phrase-starting address are passed to the next processor. The system then checks to see whether any cell used this rule during the last cycle. If the rule was used, then the rule is repeated until no processors use it. The net result is that when the longest string of adjectives preceding a noun has been parsed, the rule is used no longer and the next rule is broadcast. In this rule, any processor that contains state symbol Q1 and also contains a noun can then return a noun phrase that runs from the initial determiner to the noun itself, and thus the parsing of all such noun phrases in the system is completed.

In general, the grammar is nondeterministic; that is, there is more than one pattern for noun phrases. All such patterns are broadcast, one after another, and in the end, all the noun phrases are marked. Occasionally, one noun phrase may appear to overlap another; in that case, both can be returned or a serial computer can attempt to judge which interpretation should have priority. A typical simple scheme that does not require serial arbitration is to use the longest noun phrase if two overlap.

We have used several algorithms for assigning parts of speech to a word.¹⁴ The one currently in use depends on the scan

and sort operations described briefly in the section on "Memory-based reasoning systems" (above) and illustrated in Figure 7. Both (a) the dictionary definitions and (b) the words of text to be processed are stored, one each per virtual processor. Next, all the words, from both the dictionary and the text, are sorted. The dictionary items are, in effect, subscripted with a zero, and the words in the text are, in effect, subscripted with a one, so that at the end of the sort phase, the words appear with a dictionary entry followed by all the occurrences of that word in text, which are in turn followed by the next word in the dictionary and all its occurrences in the text. A variation of the scan operation can then be used to spread, in a time proportional to the log of the most frequently occurring words in the text, the definition to all words that follow a dictionary word. This method allows this system to assign parts of speech to 64,000 words in about 70 milliseconds.

Another scheme, currently being programmed, operates by the following method: Each dictionary word is hash coded and sent to a particular processor. To find parts of speech for words in a text, the hash code for each word of the text is computed, and a message is sent to the computed hash code address. Definitions are then sent back to all the words that requested them. The total time for this algorithm should be less than 10 milliseconds.

Other applications

Several other applications have been implemented on the CM that have application to AI research in qualitative reasoning.

- A SPICE-like VLSI-circuit-simulation system has been devised that has run a circuit of 8000 transistors. Such a system could be used as a basis for a system that could reason about the behavior or diagnosis of such circuits.

- A VLSI-layout program that uses simulated annealing has been devised. We are currently investigating the use of intelligent energy-weighting schemes that would use a database of layout decisions previously made by a designer; the intention is that the schemes assign priorities to proposed exchanges of cells.

- A system to model fluid dynamics has been built on the basis of a cellular automated model.¹⁵ The "fluid flow" operates by means of a 16-million-cell grid with four million particles moving across

it. The particles interact at each cell, and their gross behavior simulates fluid flow (including vortex shedding and aperiodic flow) without the use of Navier-Stokes equations. Such a system could also be used to make judgments about the general qualitative behavior of fluids, liquids, and articulated or flexible solids.

- Experiments have been undertaken for a variety of connectionist and other learning models (for example, Rosenberg and Sejnowski's machine).¹⁰ (See also the article by S. Fahlman and G. Hinton in this issue of *Computer*.) The Connection Machine is extraordinarily well-suited to such tasks, and early results suggest a speedup in excess of two orders of magnitude over the minicomputers and workstations on which such systems are typically run now.

- The Connection Machine system is able to analyze a pair of images of terrain, taken from two different vantage points, to determine the terrain elevation and to draw a contour map. A production-level version of the contour-mapping algorithm has been implemented and extensively tested. A typical program run processes images containing 512×512 (262,144) pixels. In such a mode, the Connection Machine system performs approximately two billion (2×10^9) operations during the inner loop of the algorithm. In the inner loop, the matchups are detected and their alignment quality is measured. This inner loop is executed in less than two seconds on a 16K CM, or in half a second on a 64K CM.

The Connection Machine system has already proved itself to be both easily programmable and applicable to a wide variety of applications. Most CM applications to date have used the method of data-level parallelism, in which each datum is assigned to a processor. Many of the applications depend on the methods of "recursive doubling," which allow associative operations (counting, addition, matrix multiplication, and so on) to be applied to large amounts of data in log time.⁸ Unlike most current commercial products, the Connection Machine scales to much larger problems by simply adding more processors, or by hooking machines together. We believe that we have only begun to explore this exciting new technology, and we look forward eagerly to the applications prospects of the next few years. □

String - Processor #	
Defined?	D
Definition Bits:	BBBB
Original-Address:	N
Black if Selected:	

The diagram shows two groups of word representations. The first group, labeled 'Text', contains five boxes for words 'the-1', 'boy-2', 'ate-3', 'the-4', and 'pizza-5'. Each box has three rows: 'D?:' followed by a binary string, 'B:' followed by a binary string, and 'N:' followed by a decimal number. The second group, labeled 'Dictionary', contains four boxes for words 'ate-6', 'boy-7', 'dog-8', and 'the-9', following the same three-row format. The 'Text' label is positioned below the first group, and the 'Dictionary' label is positioned below the second group.

the-1	boy-2	ate-3	the-4	pizza-5	ate-6	boy-7	dog-8	the-9
D?: 0	D?: 0	D?: 0	D?: 0	D?: 0	D?: 1	D?: 1	D?: 1	D?: 1
B: -	B: -	B: -	B: -	B: -	B: 100	B: 010	B: 011	B: 001
N: 1	N: 2	N: 3	N: 4	N: 5	N: -	N: -	N: -	N: -

Text

Dictionary

ate-1 ate-2 boy-3 boy-4 dog-5 pizza-6 the-7 the-8 the-9

ate-1 ate-2 boy-3 boy-4 dog-5 pizza-6 the-7 the-8 the-9

Definition not used Definition not in dictionary

Text

Dictionary

96

Acknowledgments

Credit is due especially to Craig Stanfill, who designed and implemented the document-retrieval system and who is jointly responsible with me for the idea of memory-based reasoning; to Bob Fischer, who wrote most of the report *Introduction to Data-Level Parallelism*, from which some sections of this article were adapted; to Gary Sabot, who implemented the system for the bulk processing of natural language; and to Brewster Kahle, Paul Mott, George Robertson, Shaun Keller, Sarah Ferguson, Jim Bailey, J.P. Massar, Guy Steele, Jr., Janet MacLaren, Christine Maruk, Danny Hillis, Robert Thau, Donna Fritzsche, and many others at Thinking Machines, all of whom played important roles in the work reported here.

References

1. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.
2. G. E. Blelloch, "CIS: A Massively Concurrent Rule-Based System," *AAAI-86, Proc. Fifth Nat'l Conf. Artificial Intelligence*, Philadelphia, Aug. 1986, pp. 735-741.
3. C. Stanfill and B. Kahle, "Parallel Free Text Search on the Connection Machine System," *Comm. ACM*, Vol. 29, No. 12, Dec. 1986.
4. D. C. Blair and M. E. Maron, "An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System," *Comm. ACM*, Vol. 28, No. 3, Mar. 1985, pp. 285-299.
5. Thinking Machines Corporation, *Introduction to Data Level Parallelism*, Cambridge, Mass., Apr. 1986.
6. J. R. Quinlan, "Discovering Rules from Large Collections of Examples: A Case Study," *Expert Systems in the Micro Electronic Age*, D. Michie, ed., Edinburgh University Press, Edinburgh, Scotland, 1979.
7. A. Samuel "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. Research and Development*, Vol. 3, No. 3, 1959, pp. 210-299.
8. D. Hillis and G. L. Steele, Jr., "Data Parallel Algorithms," *Comm. ACM*, Vol. 29, No. 12, Dec. 1986.
9. C. Stanfill and D. L. Waltz, "Toward Memory-Based Reasoning," *Comm. ACM*, Vol. 29, No. 12, Dec. 1986.
10. T. J. Sejnowski and C. R. Rosenberg, "NETalk: A Parallel Network that Learns to Read Aloud," tech. report JHU/EECS-86-01, The Johns Hopkins University, Electrical Engineering and Computer Science Technical Reports, 1986; contact Terrence Sejnowski, Department of Biophysics, Johns Hopkins University, Baltimore, MD 21218.
11. R. S. Michalski, J. Carbonell, and T. Mitchell, *Machine Learning*, Tioga Publishing Co., Palo Alto, Calif., 1983.
12. M. Lebowitz, "Categorizing Numerical Information for Generalization," *Cognitive Science*, Vol. 9, No. 3, July-Sept. 1985, pp. 285-308.
13. W. A. Woods, "Transition Network Grammars for Natural Language Analysis," *Comm. ACM*, Vol. 13, No. 10, Oct. 1970, pp. 591-606.
14. G. Sabot, "Bulk Processing of Text on a Massively Parallel Computer," *Proc. 24th Annual Meeting of the Association for Computational Linguistics*, New York, June 10-13, 1986, pp. 128-135.
15. J. Salem and S. Wolfram, "Thermodynamics and Hydrodynamics with Cellular Automata," in *Theory and Applications of Cellular Automata*, S. Wolfram, ed., World Scientific, 1986.



David Waltz is a senior scientist at Thinking Machines Corporation and a professor of computer science at Brandeis University. He received a PhD from MIT in electrical engineering and artificial intelligence in 1972, and also holds an MS and a BS from MIT—both in electrical engineering. Until 1984, he was professor of computer and electrical engineering at the University of Illinois. He is editor of the journal *Cognitive Science*, and is a former chairperson of the ACM Special Interest Group on Artificial Intelligence (SIGArt). Waltz's interests include constraint-propagation models for computer vision and other artificial intelligence tasks, natural-language processing, massively parallel connectionist models of cognition, and practical applications of artificial intelligence.

Readers may write to David L. Waltz at Thinking Machines Corp., 245 First St., Cambridge, MA 02142-1214.

IEEE ICC' 87

Workshop on the Integration of Expert Systems into Network Operations Seattle Sheraton June 11-12, 1987

sponsored by

IEEE Communication Society
The Committee on Network Operations and Management
and co-ordinated with the

IEEE International Conference on Communications '87 (ICC)

The workshop is aimed at Network Operations planners, developers, managers, and Expert System developers; it will explore the issues that must be addressed to integrate the Expert System Technology into the telecommunications network operations and management. The emphasis will be on knowledge engineering and prototyping experience gained in deploying the Expert Systems. The highlights of the workshop sessions are as follows:

June 11	Morning:	A tutorial on Expert System Technology with emphasis on Network Operations.	June 12	Morning:	Invited papers, talks, and a panel discussion on promise of the Expert System Technology in the Network Operations.
	Afternoon:	Knowledge engineering issues and case studies of Expert Systems in Network Operations.			

The last date for registration is March 31, 1987. For more information and registration contact:

Nassem A. Kahn, Technical Chairperson
AT&T Bell Laboratories
HR 1 J-220
480 Red Hill Road
Middletown, NJ 07724
(201) 758-5590

Rao V. Mikkilineni, Workshop Chairperson
Bell Communications Research
NVC 2Z-155
331 Newman Springs Road
Red Bank, NJ 07701
(201) 578-5590

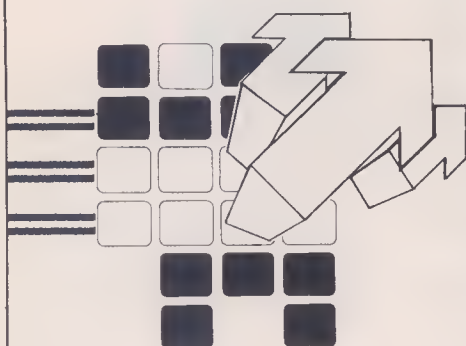
Workshop Committee includes:

J. J. Bodnar, AT&T Bell Labs; D. J. Sassa, Bell Communications Research; A. Schwartz, AION; P. M. Bauer, U. S. West; E. D. Sacerdoti, Teknowledge; Y. V. R. Reddy, West Virginia University; D. A. Rosenzweig, NYNEX; J. Kowalik, Boeing Computer Services; A. Croisier, IBM Research, France.



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.

ADVANCE PROGRAM



Sixth Symposium on Reliability in Distributed Software and Database Systems

SPONSORS

Computer Society of the IEEE 
TC on Distributed Processing
TC on Fault-Tolerant Computing

March 17-19, 1987

Hilton Conference Center
Kingsmill — Williamsburg, Virginia

IN COOPERATION WITH

ACM 
NASA-Langley Research Center
College of William & Mary, Computer Science Department
Williamsburg, Virginia
IEEE — Tidewater section

THEME: The symposium addresses all aspects of reliable software with special emphasis on database systems. The program presents a new tutorial on modelling software reliability, a most critical aspect in the software development process. The papers cover reliability in the critical areas of communications, environments, structures and replicated data, synchronization and algorithms. The keynote and invited speakers provide new results and status in these important areas. The conference is designed for effective communication between participants and attendees to penetrate and exchange the latest problems and results.

PROGRAM

■ Tuesday, March 17, 1987 9:00-17:00

Tutorial — Modelling Software Reliability, Amrit L. Goel, Syracuse U.

■ Wednesday, March 18, 1987

8:00-9:00: Registration

9:00-9:30: Opening Ceremony

9:30-10:30: Keynote Address — Distributed Operating Systems:

Andrew S. Tanenbaum, Vrije U., Amsterdam, NL

10:30-11:00: Break

11:00-12:30: Session I — Communication and Control:

S.K. Shrivastava, Newcastle-Upon-Tyne, UK

• A Reliable and Secure UNIX Connection Service: D. Draheim,

B. Miller, S. Synder, U. Wisconsin

• Preserving Context Information on an IPC Abstraction:

L. Peterson, U. Arizona

• Adaptable, Reliable Software for Distributed and Parallel Real-time

Systems: K. Schwan, T. Bihari, B. Blake, Ohio State U.

12:30-14:00: Lunch

14:00-15:30: Session II — Programing Support: R. Schlichting, U. Arizona

• Reliability in Distributed Programing Environments: G. Kaiser, Columbia U., Kaplin, Illinois

• Bugnet-A Real-time Distributed Debugging System: S. Jones, R. Barkan, L. Wittie, SUNY-Stonybrook

• DTM — A Method for Testing Distributed Systems: D. Haban, U. Kaiserslautern, FRG

15:30-16:00: Break

16:00-17:30: Session III — Replicated Data: Carla Ellis, Duke U.

• On Improving the Availability of Replicated Files: D. Long, J. Paris, U. Calif.-San Diego

• Access Control in a Heterogeneous Distributed Database

Management System: C. Wang, D. Spooner, Rensselaer Tech

• Independent Recovery in Distributed Systems: G. Chang, M. Gouda, U. Texas-Austin

18:30-19:30: Reception

■ Thursday, March 19, 1987

8:30-10:00: Session IV — Object Based Systems: Robert Cook, U. Virginia

• Exploiting Type Inheritance Facilities to Implement Recoverability in Object Based Systems: G. Dixon, S. Shrivastava, Newcastle-Upon-Tyne, UK

• Fault Tolerant Computing in an Object Based Distributed Operating

System: M. Ahamad, P. Dasgupta, R. Leblanc, C. Wilkes, Georgia Tech

• Design and Implementation of Nested Transactions in Eden:

C. Pu, J. Noe, U. Washington

10:00-10:30: Break

10:30-12:00: Session V — Concurrency & Synchronization: K. Birman, Cornell U.

• Synchronization of Redundant Computation in a Distributed System: N. Natarajan, T. Jian, Penn. State U.

• Fault Masking and Sequence Agreement by a Voting Protocol with Low Message Number: K. Echite, U. Karlsruhe, FRG

• Performance Characterization of Quorum-Consensus Algorithms for Replicated Data: M. Ahamad, M. Ammar, Georgia Tech.

12:00-14:00: Lunch — Invited Address — Nondeterminism as Model for Faulty Systems: K. Mani Chandy, U. Texas-Austin

14:00-14:15: Break

14:15-15:45: Session VI — Algorithms: Ray Strong, IBM, Almaden Labs

• Maintaining Availability of Replicated Data in the Dynamic Failure Environment: D. Barbara, U. Simon Bolivar, Ven.; H. Garcia-Molina, B. Kogan, Princeton U.

• A Gracefully Degradable Algorithm for Byzantine Agreement:

F. DiGiandomancio, M. Guidotti, F. Grandoni, E. Simoncini, IRI-CNR, Pisa, IT

• Early Stopping Algorithm for Distributed Agreement Under Fail-stop, Omission & Timing Fault Types: P. Ezhilchilvan, Newcastle-Upon-Tyne, UK

Sixth Symposium on Reliability in Distributed Software and Database Systems

REGISTRATION FORM

Symposium on Reliability in Distributed Software and Database Systems

Complete and return this form with check made payable to:

Distributed Software Symposium
c/o Department of Computer Science
College of William and Mary
Williamsburg, VA 23186

Advance Registration (prior to 2/27/87)

	Member	Non-member	Student/retired
Conference	\$110	\$140	\$15
Tutorial	\$110	\$140	\$110

Late Registration (after 2/27/87)

Conference	\$150	\$175	\$20
Tutorial	\$150	\$175	\$150

Conference amount \$ _____

Tutorial amount \$ _____

TOTAL ENCLOSED \$ _____

IEEE or ACM Membership No. _____

Name _____

Organization _____

Street address _____

City/State/Zip _____ Telephone _____

Credit card name _____

Card no. _____

Expiration date _____

Cardholder's name _____

Signature _____

To guarantee your reservation, either a check or appropriate credit card information (VISA, MasterCard, Choice, must be included with the registration form.

NOTES: Registration fee includes admission to the technical sessions, one copy of the symposium proceedings (except students), break refreshments, lunch both days, and reception. Refunds (less \$15 handling fee) will be made if request is received in writing by March 6, 1987. Tutorial registration includes tutorial notes, break refreshments, and lunch.

HOTEL RESERVATION FORM

Symposium on Reliability in Distributed Software and Database Systems

Hotel Address:

Williamsburg Hilton & National Conference Center
50 Kingsmill Road
Williamsburg, VA 23185
(804) 220-2500

The hotel is located 3 miles east of Williamsburg on US route 60. Williamsburg is accessible from Newport News (20 mi), Richmond (70 mi), or Norfolk (45 mi) airports via I-64.

Rates

Single or with spouse \$65 + 6% tax
Double (2 attendees) \$80 + 6% tax

All major credit cards accepted.

by February 14, 1987. To guarantee reservations, we must receive one night's deposit including tax.

Reservations made after February 14, 1987 are subject to space available. Check-in time is 3:00 pm. Check-out time is 12:00 noon. Cancellations must be received by the hotel at least 24 hours prior to date of arrival.

Name _____

Street address _____

City/State/Zip _____ Telephone _____

☐ Single or with spouse

☐ Double (two attendees)

Name of other attendee _____

Arrival date/time _____ Departure date _____

Credit card name _____

Card no. _____

Expiration date _____

Cardholder's name _____

Signature _____

Connectionist Architectures for Artificial Intelligence

Scott E. Fahlman and Geoffrey E. Hinton

Carnegie-Mellon University

Massively parallel networks of simple neuron-like processing elements may hold the key to some important aspects of intelligence not captured by existing AI technology on serial machines.

Current AI technology can do a good job of emulating many of man's higher mental functions, but some of the most fundamental aspects of human intelligence have proven more elusive. AI can match the best human experts on certain narrow technical problems, but it cannot begin to approach the common sense and sensory abilities of a five-year-old child. Some important ingredients of intelligence seem to be missing, and our technology of symbolic representation and heuristic search, based on serial computers, does not seem to be closing the gap. Among the missing elements are the following:

- The human memory can store a huge quantity and variety of knowledge, and can find relevant items in this storehouse very quickly and without apparent effort. The phenomenon we call common sense is complex, but it derives in part from the ready availability of a large body of assorted knowledge about the world. Our serial machines can store large amounts of information, but it is very hard to make this knowledge an effective part of the machine's activities.

- In many domains, human recognition abilities far exceed what our machines can accomplish. Whether the domain is speech recognition, vision, or some higher-level task like medical diagnosis, the key operation seems to be an ability to locate, from among all known candidates, the one that best matches the sample to be identified. We humans can do this even with noisy, distorted input data and faulty expectations. For us, this is quick and seemingly effortless.

- In many cases, humans seem to handle information in some form other than the symbolic assertions of traditional AI. We can all recognize an elephant, but few of us can describe its appearance symbolically in an unambiguous way. We have difficulty coming up with formal symbolic descriptions for movements, shapes, sounds, and spatial relationships, and yet people work easily in all these domains. It can be argued that we humans use internal symbolic representations that we cannot access consciously, but it seems more plausible that some other kinds of representations are in use.

The traditional AI approach to knowledge and recognition problems is to use ever more complex and clever strategies to reduce the need for excessive search and computation. An alternative is to solve such problems with a less complicated but very cycle-intensive approach, using very large numbers (millions) of very simple processors to get the job done in a reasonable time. For example, one approach to interpreting visual input is to use clever reasoning to restrict the areas in which a computationally expensive edge-finder is applied. The alternative is to simply accept the cost of doing high-quality edge finding all over the image—even in places that will turn out not to be critical to later stages of interpretation.

A number of researchers have begun exploring the use of massively parallel architectures in an attempt to get around the limitations of conventional symbol processing. Many of these parallel architectures are connectionist. The system's collection of permanent knowledge is stored

as a pattern of connections or connection strengths among the processing elements, so the knowledge directly determines how the processing elements interact rather than sitting passively in a memory, waiting to be looked at by the CPU. Some connectionist schemes use formal, symbolic representations, while others use more analog approaches. Some even develop their own internal representations after seeing examples of the patterns they are to recognize or the relationships they are to store.

Connectionism is somewhat controversial in the AI community. It is new, still unproven in large-scale practical applications, and very different in style from the traditional AI approach. We have only begun to explore the behavior and potential of connectionist networks. In this article, we describe some of the central issues and ideas of connectionism, and also some of the unsolved problems facing this approach. Part of the motivation for connectionist research is the possible similarity in function between connectionist networks and the neural networks of the human cortex, but we concentrate here on connectionism's potential as a practical technology for building intelligent systems.

What is connectionism?

Jerry Feldman coined the term "connectionism" to refer to the study of a certain class of massively parallel architectures for artificial intelligence. A connectionist system uses a large number of simple processing elements or *units*, each connected to some number of other units in the system. The units have little information stored internally, typically only a few *marker bits* or a single scalar *activity-level*, used as a sort of short-term working memory. The long-term storage of information is accomplished by altering the pattern of interconnections among the units, or by modifying a quantity called the *weight* associated with each connection. This use of connections, rather than memory cells, as the principal means of storing information motivated the name connectionism.

The parallel processing units in a connectionist network do not follow individual programs. The units are capable of only a few simple actions such as accepting incoming signals, performing some Boolean or arithmetic processing on the data, and sending signals out over some or all of the

connections. These operations may be completely autonomous—part of a unit's built-in behavior—or they may be controlled by commands broadcast by some external controller, perhaps a serial computer of the traditional kind.

Since all of the connections can carry signals simultaneously, and all of the processing elements can act in parallel to integrate their arriving data, a connectionist system can bring a large amount of knowledge to bear simultaneously when making a decision, and can weigh many choices at once.

In some connectionist systems, the parallelism is used to implement a sort of simultaneous brute-force search through units, each representing a single item in the knowledge base. In other systems, the parallelism is used to allow richer representations; the pattern of activity in a large group of units represents an item, and different items are represented by alternative patterns of activity. Given some inputs and an initial state of the network, one of these patterns will emerge. Many connections, representing many small pieces of knowledge, will play a role simultaneously in determining which alternative will win. Whichever strategy is used, this ability to bring a lot of knowledge into the game at once is a major reason for the growing interest in connectionism.

The interunit signals sent through the connections are typically single-bit markers or continuous scalar values. Thus, we speak of *marker-passing* or *value-passing* parallelism. A unit can receive many of these signals at once, each arriving over a different connection. These multiple signals are combined upon receipt: Multiple instances of the same marker bit are simply OR'ed into one, and multiple scalar values are usually just combined into a weighted sum as they arrive at the destination unit. Systems that send more complex symbolic messages from unit to unit, the so-called *message-passing* parallel systems, are not usually considered connectionist systems because they require much more complex processing units with a considerable amount of storage for the messages.

An important difference between the connectionist approach and the more conventional "modestly parallel" architectures for AI is that the connectionists are willing to assign a processing element to each tiny subtask (one element for every item of knowledge in the system, for example) and to postulate that there are enough of these simple processing elements to handle the task at hand. The

more conventional approach assumes a fixed number of larger processors (typically between 2 and 1024) and tries to find ways of cutting the problem up into that many pieces, all of which can be worked on concurrently.

In a serial system, the time required to sift through a finite set of items in memory or to consider a finite set of hypotheses in a recognition task grows linearly with the size of the set. The modestly parallel approach attempts to approach N -fold speedup from N processors. The connectionist thinks in terms of performing these simple tasks in constant time, while the amount of hardware grows linearly with the number of memory items or the number of hypotheses to be considered at once. This view may seem less radical if we think of a connectionist unit not as a CPU but as a fancy kind of memory cell that stores knowledge in its connectivity or connection weights. We need enough of these memory cells to hold the system's knowledge.

Many kinds of connectionist architectures are being investigated by the small but growing community of researchers interested in this kind of parallelism. It is hard to say much more about these systems as a class, and in an article of this length it is impossible to mention all of the connectionist research going on. We will describe some of the key issues and ideas that seem important to us, and mention one or two pieces of work exemplifying each of these key ideas.

Distributed representations

The simplest way to represent things in a massively parallel network is to use *local representations*, in which each concept or feature is represented by a specific piece of hardware. For example, when the system wants to work with the concept of "elephant," it turns on the elephant unit. This kind of representation is easy to create and easy to understand. Unfortunately, if the elephant unit breaks, the system loses all of the knowledge tied to it. This creates some obvious reliability problems: In a system with millions of processing elements, not all of them will be working all the time. It also means that most of the units will be idle most of the time.

Many neuroscientists believe that the brain does not work this way, that instead it uses some sort of *distributed representa-*

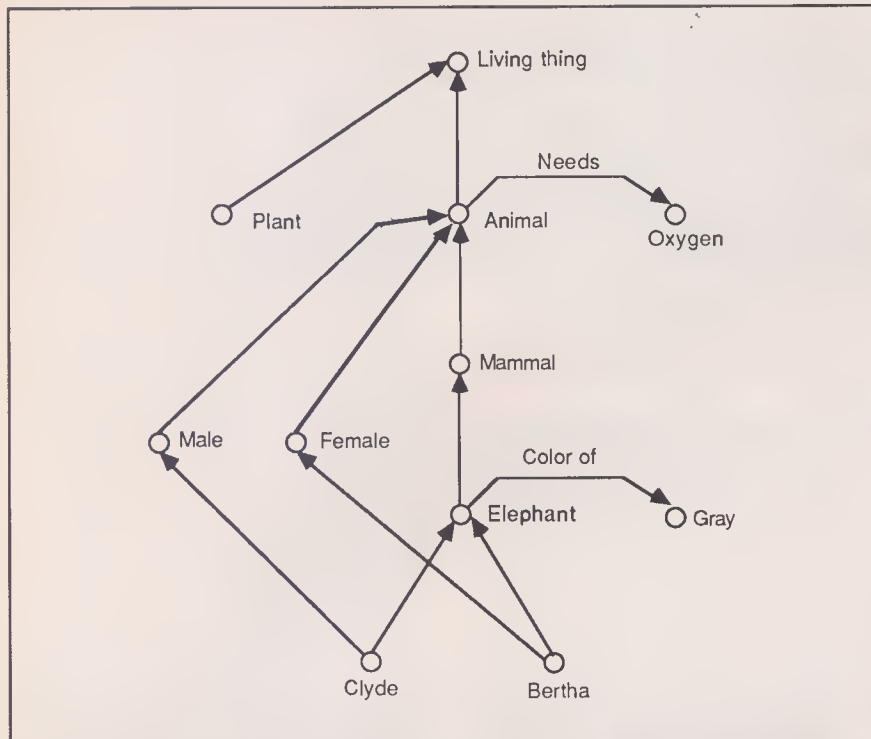


Figure 1. A portion of a NETL semantic network. This fragment describes Clyde the elephant and related information. (Source: *Artificial Intelligence*, Vol. 1, 1985. Used by the courtesy of MIT Press, publisher.)

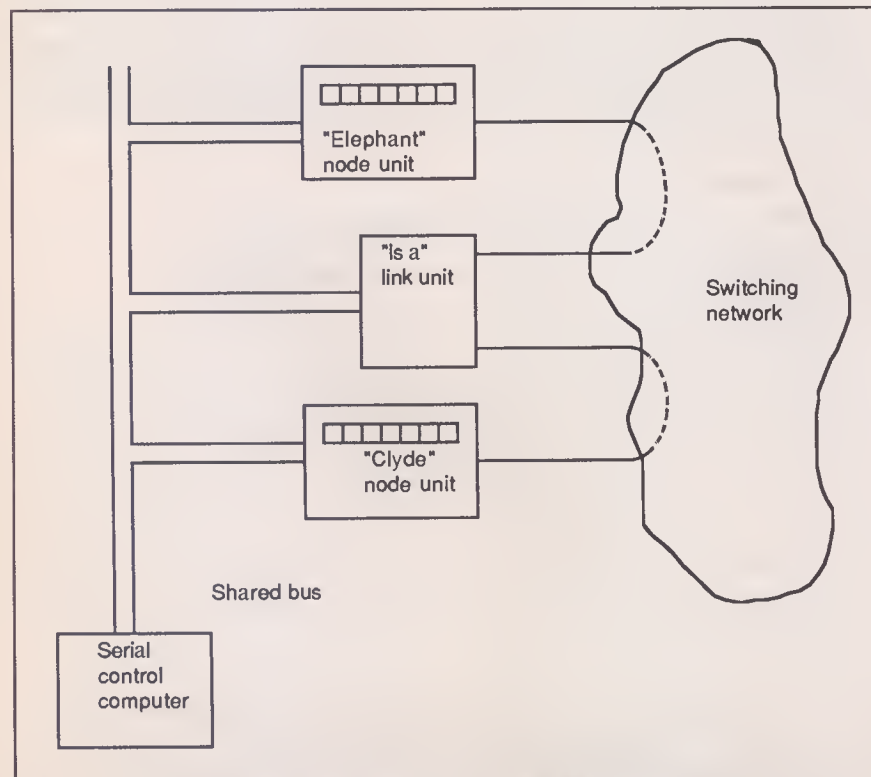


Figure 2. NETL hardware corresponding to "Clyde is an Elephant." (Source: *Artificial Intelligence*, Vol. 1, 1985. Used by the courtesy of MIT Press, publisher.)

tion: A concept like "elephant" is represented not by a single neuron, but by a pattern of activation over a large number of neurons. They often use the analogy of a hologram, in which each point of the image is constructed from information from all over the film; if some part of the hologram is destroyed, the total image is degraded slightly, but no part of the image is completely lost. A distributed representation in a massively parallel network (whether built from neurons or silicon) would have a similar kind of reliability: If a few of the units malfunction, the resulting pattern is imperfect but still usable. Each macroscopically important behavior of the network is implemented by many different microscopic units, so any small random subset can disappear without changing the macroscopic description of the network's behavior.

This kind of inherent fault-tolerance has important implications for the construction of large-scale parallel networks. For example, wafer-scale integration becomes more feasible, since a few malfunctioning units in the wafer would simply be ignored. It is almost certainly easier to build a billion-transistor system, in which only 95 percent of the circuit elements have to work, than to build a million-transistor system that has to be perfect. It can also be shown¹ that in many situations it is much more efficient, in terms of the number of units required, to represent a value using a *coarse-coded* distributed scheme than to assign a single unit to represent each small interval in the range. In the coarse-coded scheme, each unit is coarsely tuned to cover a considerable range of values, and a particular value is represented by activity in a number of overlapping units.

The main disadvantage of distributed representations is that they are hard for an outside observer to understand or to modify. To add a single new piece of macroscopic knowledge to a network that uses distributed representations, it is necessary to change the interactions between many microscopic units slightly so that their joint effects implement the new knowledge. For any problem of significant size, it is nearly impossible to do this by hand, so it follows that a network using a distributed representation must employ some sort of automatic learning scheme. In the absence of an automatic learning procedure that works efficiently in large networks, distributed representations are generally too awkward to be of much practical value.

NETL: A connectionist system for symbolic knowledge

The NETL system² is an example of a connectionist architecture that uses local representations. It was designed to store and access a large number of symbolic assertions, and to perform certain simple searches and deductions within this collection of knowledge. NETL is an implementation in hardware of a *semantic network*, a graph-like data structure in which the nodes represent noun-like concepts and the links represent relationships between these concepts. Such a network is illustrated in Figure 1; the NETL hardware corresponding to a part of this network is illustrated in Figure 2.

In the NETL system, each of the nodes in the network is represented by a simple processing unit, capable of storing a few single-bit markers and of performing simple Boolean operations on the markers. Each link is also a simple processing unit, wired up to two or more node units. Link units, too, can perform simple Boolean operations, which generally amount to passing a specific marker from one of the attached nodes to another. All of the nodes and links in the network can perform these operations simultaneously in response to commands broadcast by the system controller, a serial machine of the familiar sort.

Every time a new assertion is added to the system, new nodes and links must be wired into the network to represent that assertion. Since all the node-to-link connections must be capable of carrying signals simultaneously, they must be true private-line connections and not just addresses sent over a shared party-line bus. Of course, in a practical implementation, these new connections would be established in a switching system resembling a telephone exchange and not by stringing tiny wires from unit to unit.

A NETL network can perform searches and simple inferences that go beyond what can be done with a simple associative memory. For example, a very heavily used operation in most AI knowledge-based systems is *inheritance*: We want anything we know about the typical elephant to apply to the subclasses and individuals below elephant in the hierarchy of is-a links (unless that inherited information is specifically canceled for some individual). Because of branching in the is-a hierarchy, an individual node like clyde might inherit in-

formation from a large number of superior nodes, so we need some very fast way of scanning a large part of the network.

In NETL, inheritance is handled by marker propagation. If the problem is to find the color of Clyde the elephant, the controller sets marker 1 on the clyde node, then repeatedly orders any is-a link with marker 1 set on the node below it to pass it on to the node above. When the network settles, we have marker 1 on all of the nodes representing superior classes from which clyde is supposed to inherit properties. We then command every color-of link with marker 1 on its tail to put marker 2 on its nose. Finally, we ask every node marked with marker 2 to report its identity to the controller. In this case, we get a single winner, the gray node. In a few cycles, we have followed a chain of inference to locate this information, regardless of where in the network the color information may have been attached. (The actual sequence of operations is slightly more complex than this, due to the possible presence of exceptions in the network, but the idea is the same.)

Because of its ability to do set intersections in parallel, NETL can handle recognition tasks of a certain limited kind. Suppose we are confronted with a gray four-legged mammal and we want to quickly locate any stored description that exhibits the intersection of these features. By marker-propagation from the feature nodes, we can mark all of the gray things with one marker, four-legged things with a second marker, and mammals with a third. Then we simply broadcast a command that any unit with all of these markers should queue up to report its identity to the controller. The intersection step takes only a small, constant time, regardless of the size of the sets being intersected.

This approach to recognition is very strong in some ways, but weak in others. On the positive side, NETL examines all of the descriptions in memory at once, not depending on heuristics that might prematurely rule out the right answer. Any set of observed features that is sufficient to select one of the stored descriptions will suffice, and if not enough features are present, the system will still narrow the set of possibilities down as much as possible. When new descriptions are added to the network, the knowledge immediately becomes an effective part of the recognition process. We do not need to hire a programmer or knowledge engineer to specify exactly how and when each piece of knowledge is to be applied. On the negative side, NETL treats

every feature as an atomic entity that is either present or absent and expects that the winning description will explain all of the observed features. Like traditional symbolic AI systems, NETL works best in clean domains, far from the noise and confusion of the low-level sensory inputs.

The NETL architecture has been simulated, but not yet implemented directly. The node and link units are so simple that several thousand of them could fit on a single chip. The hard problem is to design the switching network that implements the node-to-link interconnections. A design study by Fahlman³ demonstrated that a NETL machine can be built for only a few times the cost of a conventional memory system able to hold the same amount of information. The Connection Machine⁴ was designed in part to implement a NETL-like knowledge base, though the hardware is general enough to do other jobs as well. The current 64,000-processor version of this machine is large enough to handle a substantial body of knowledge, by the standards of current AI systems. The proposed million-processor version may be able to handle enough assorted knowledge to exhibit some degree of common sense.

Layered value-passing networks for recognition

As we mentioned earlier, a marker-passing system like NETL does not gracefully handle the messy recognition problems we encounter in the real world. Whether we are dealing with vision, speech understanding, medical diagnosis, or some other real-world recognition problem, we normally face a situation where no match is ever perfect and in which few of the incoming features are noise-free and certain. Some features provide strong evidence for particular hypotheses; other features are only suggestive. Some features are clearly present; others are borderline cases or the result of low-confidence observations. Instead of the all-or-none set operations of NETL, we need a system that can combine many observations of varying quality and find hypotheses that fit well, even if they do not match perfectly.

This kind of problem can be handled by a *value-passing* system, in which each connection has an associated scalar weight. Each unit computes a weighted sum of the incoming values and passes this sum through a nonlinear function, with the

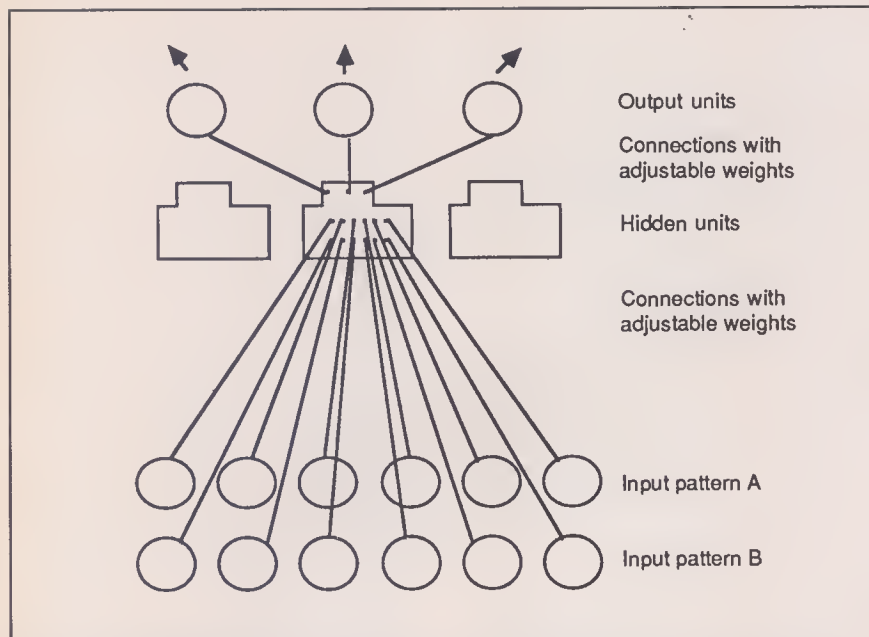


Figure 3. Value-passing network to compare two six-bit patterns. Input patterns A and B are identical, except that pattern A may be rotated one position to the left or right from pattern B. The three output units represent a left rotation, no rotation, and a right rotation, respectively. This task cannot be done without some hidden units. In this network we employ 12 hidden units, though only three appear in the diagram. An appropriate set of weights can be learned using the back-propagation method.

resulting value becoming the unit's output. A value-passing network may be implemented in either digital or analog hardware, but computationally it can be viewed as a sort of analog computer.

Using these value-passing units, we can set up a layered recognition system able to handle uncertain observations and varying degrees of evidence. Each directly observable feature is represented by an *input unit* whose value represents the probability that a discrete feature is present, the magnitude of a continuous quantity (like intensity), or the probability that a magnitude lies within a particular interval. Each of the possible hypotheses that we wish to evaluate is represented by an *output unit*.

In the simplest of these networks, the input units are connected directly to the output units by a set of connections with modifiable weights. In such networks, it is not necessary to set the connection weights by hand. Given a set of input vectors and the desired output for each, the perceptron convergence procedure⁵ can be used to find a set of weights that will perform this mapping, if such a set exists.

Unfortunately, for most interesting recognition tasks there is no set of weights in a simple two-layer network that will do the

job. In most cases, one cannot treat the lowest-level features as independent sources of evidence and simply add them up. It is necessary to introduce one or more intervening layers of hidden units, which combine the raw observations into higher-order features more useful in determining the output. Consider, for example, the problem of deciding whether an image contains a telephone. The fact that a single pixel in the image has high intensity cannot be used directly as evidence for or against a telephone, because telephones can be black or white. It is necessary to extract relationships between pixel intensities (such as edges) before trying to detect the telephone.

Figure 3 shows a simple task for which hidden units are essential. Pattern A is identical to pattern B, except that it may be rotated one step to the left or right. The task is to determine the shift for any pattern of input bits. This cannot be done without hidden units because each input bit, considered in isolation, provides no evidence whatsoever about the shift. All the information is in the joint behavior of combinations of input bits, so one or more layers of hidden units must be used to extract informative multi-unit combina-

tions, which then can be combined into an overall answer.

Learning which of the exponentially many possible combinations are relevant for predicting the output is a hard problem. Techniques such as back-propagation (described below) can indeed discover a set of weights and hidden-unit assignments that, taken together, will solve the problem. Figure 4 shows some examples of hidden units and associated weights taken from one such solution.

Minsky and Papert⁶ thoroughly analyzed which tasks require hidden units and demonstrated that an insightful way of categorizing tasks is in terms of how many input units must connect to each hidden unit in a three-layer network (this determines the order of the statistics that can be extracted). Unfortunately, they gave no procedure for learning appropriate hidden units, and suggested that there may not be any simple general procedure.

For many years after Minsky and Papert's book was published, AI researchers criticized the connectionist approach because the effective learning procedures were restricted to networks incapable of performing tasks like figure-ground segmentation or viewpoint-independent shape recognition. More recently, a deeper appreciation of the sheer quantity of computation required and a better understanding of the nature of the computations has led to renewed interest in massively parallel approaches to perception.⁷

Learning representations

Three-layered nets of the kind studied by Minsky and Papert do not have the freedom to choose their own representations because the weights between the input units and the feature detectors in the middle layer are predetermined and do not learn. If we allow these weights to learn, the network can decide for itself what the units in the middle layer will represent. Consequently, we get a much more interesting and powerful kind of learning, especially when extended to multiple hidden layers. However, finding learning procedures that choose good representations is difficult, and because the space of possible representations is very large, any procedure that explores the space in a random way will be very slow.

One promising new procedure, called back-propagation, has been discovered in-

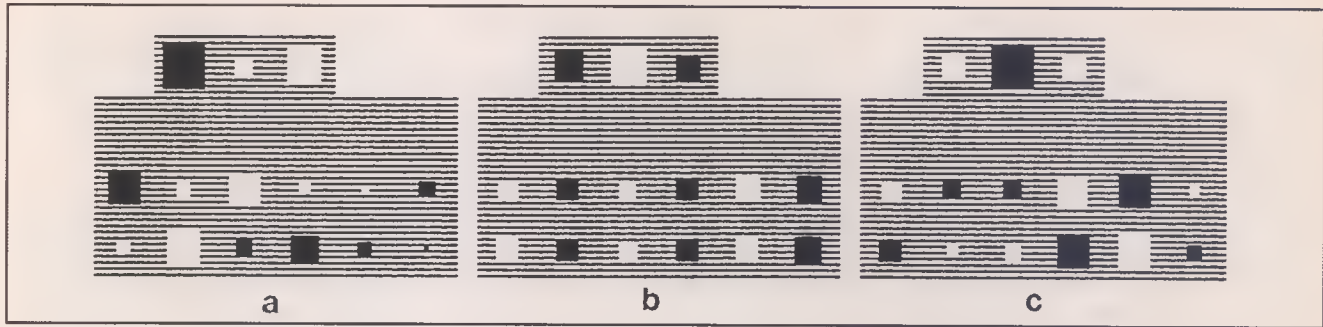


Figure 4. The weights learned by three of the twelve hidden units. The white squares are positive weights, the black squares are negative weights, and the area of a square is the magnitude of the weight. The bottom two rows in each hidden unit show the weights coming from the input units, and the top row shows the weights to the three output units that represent the three possible shifts. All the units shown learned positive thresholds (not shown here) so that they will not come on unless they receive a net positive input. Unit a detects some cases of right-shift; the two large negative weights prevent it from responding to left shifts. Unit b detects some no-shift cases. Unit c responds positively to shifts in either direction, but not to no-shift cases.

dependently by David Rumelhart, David Parker, and Yann Le Cun. It involves two passes each time an input vector is presented. In the forward pass, activity starts at the input units and passes through the layers to produce an output vector. In the backward pass, the derivative of the error (the difference between the actual output vector and the desired output vector) is back-propagated through the same connections but in the reverse direction. This allows the network to compute, for each weight, the gradient of the output error with respect to that weight. The weight is then changed in the direction that reduces the error. So learning works by gradient descent on an error surface in weight space.

Back-propagation has already been shown capable of learning many different kinds of interesting representation in the hidden units. It can learn optimal codes for squeezing information through narrow bandwidth channels, or sets of optimal filters for discriminating between very similar noisy signals.

Sejnowski and Rosenberg⁸ showed that a back-propagation net can be trained to transform an input vector that encodes a sequence of letters into an output vector that encodes the phonemic features of the sequence; that phonemic output can then be used to drive a speech-synthesis device. From the input-output examples it sees during training, this system is able to extract both the regularities exhibited by the mapping, such as the effect of a terminal "e" in changing the sound of the preceding vowel, and specific exceptions, such as the odd pronunciation of a word like "women." This same task is performed in commercial speech-generation systems by

conventional programs of considerable size and complexity.

Back-propagation can also be used to learn the semantic constraints that underlie a set of facts. One five-layer network⁹ was trained on a set of 100 triples such as (Victoria has-father Christopher) or (Christopher has-wife Penelope) derived from two family trees involving people of two nationalities. The input vector represented the first two terms of a triple and the required output vector represented the third term. As far as the network was concerned, these input vectors were just arbitrary symbols, but after extensive training the network could generalize appropriately to triples on which it had not been trained, like (Victoria has-mother ??). By recording the set of triples for which each of the hidden units became active, it was possible to show that these units had learned to represent important properties like "Italian" or "old" that were never mentioned in the input and output.

Back-propagation does gradient descent in the space of possible representational schemes, and these semantic features happen to be a very good representation for the relationships the network is trained on. The hidden features and their interactions encode the underlying regularities of the domain. When viewed locally, the learning procedure just tunes the weight parameters, but the global effect is that the network does structural learning, creating new terms that allow it to express important regularities. This example also illuminates the "local versus distributed" issue. The network's internal representation of each person is a distributed pattern of active semantic features, but each of these local features captures an important

underlying regularity in the domain.

Back-propagation has similarities to the Baum-Welch algorithm for tuning parameters in a stochastic, finite-state automaton. These trainable automata are widely used as generative models in practical speech-recognition systems¹⁰ and there is some hope that procedures like back-propagation will be able to improve on these models by overcoming an important limitation. In a finite-state automaton, the system's knowledge of what it has produced so far is encoded as the current node in a graph of states and transitions. So if the automaton needs to remember 20 bits of information about the partial sequence to constrain the rest of the sequence, it requires at least 2^{20} nodes. In a connectionist network, many units can be active at once, so the number of units need not grow exponentially with the amount of information that must be carried along in generating a sequence.

Another interesting new learning procedure, called A_{R-P} , was described by Andy Barto.¹¹ He showed how layered networks of relatively simple stochastic units can learn to cooperate in order to maximize a global payoff signal that depends on the output vector and is received by every unit. Like back-propagation, this procedure learns interesting internal representations, but it is slower because it discovers the effect of changing a weight by sampling the effects of random variation instead of by explicit computation of the gradient. The A_{R-P} model is exciting as a biological model because it does not need a separate pass in which to back-propagate detailed error information.

A major criticism of all the current multilayer learning procedures is that they are

slow even for modest examples and they appear to scale poorly. The obvious problem of gradient descent—that it will get stuck in a poor local minimum—turns out to be only a minor problem in practice. The real difficulty is that simple gradient descent is very slow because we have information about only one point and no clear picture of how the surface may curve. In high-dimensional spaces the error surface usually contains ravines that are highly curved across the ravine and slope gently downwards along the ravine. Small steps take forever to meander down the ravine, and big steps cause divergent oscillations across the ravine. Even with a large speed-up from faster implementation technology, these learning techniques are too slow to handle many problems of interest. Future progress may well depend on discovering ways of partitioning large networks into relatively small modules that can learn more or less independently of each other.

Constraint-satisfaction in iterative networks

The idea that perceptual interpretation consists of transforming an input vector through successive layers of units until it becomes a categorization is rather restrictive. Most real recognition tasks require that multiple layers of features be recognized all at once and that the result of perception be a coherent, articulated structure rather than a single category. In visual recognition, for example, we might have a scene, various objects in the scene, their parts, and subparts, all constraining one another and all relating in complex ways to the low-level stimuli the system is receiving. An African context provides evidence in favor of seeing an elephant. An elephant helps us to identify an African scene, and that in turn can help us to spot the giraffe. The trunk, tusks, and ears all are features that help us to recognize an elephant. Likewise, knowing that we're looking at an elephant helps us to recognize the elephant's parts.

In a situation like this, we want to start with a network that records the tangle of interlevel constraints and evidence relationships, put in our observations as a sort of boundary condition, and get the system to settle into the best possible solution (or set of solutions) spanning all the levels of description. Some constraints and expectations will be violated, but we want to

find the best-scoring constellation of decisions overall.⁷

The task is similar to finding the solutions to a set of simultaneous equations. A value-passing network would seem a sensible parallel way of converging on a good solution—a sort of analog computer with links representing the constraints. Non-learning-constraint networks of this sort have been studied for a variety of problem domains, most notably at the University of Rochester.¹² In some simple cases, the network can be guaranteed to settle to the best final state no matter what values it starts with, but for more difficult problems, especially ones that involve discrete decisions, a value-passing network with loops and many nonlinear decision elements is generally not well-behaved. Such networks tend to oscillate or to get trapped in uninteresting states that do not represent good solutions to the problem.

Hopfield and Boltzmann networks for constraint satisfaction

One way to guarantee that a network will settle down is to show that there exists some cost function that decreases every time one of the values changes. Hummel and Zucker¹³ showed that there exists such a function for networks that pass values and have symmetrical connections. At about the same time, Hopfield¹⁴ identified a cost function (which he called "energy") for networks of symmetrically connected binary threshold units.

Hopfield's energy function can be interpreted in the following way: A connection between units i and j with a positive weight represents a constraint that if one of these units is on, the other one should be also. A negative connection weight says that if one of the units is on, the other should not be. The weight corresponds to the penalty to be applied if the constraint is violated. The energy of any state of the network is given by

$$E = -\sum_{i < j} s_i s_j w_{ij} + \sum_i s_i \Theta_i$$

where w_{ij} is the weight on the connection between units i and j , s_i is 1 if unit i is on and 0 otherwise, and Θ_i is a threshold for unit i .

Given Hopfield's quadratic definition of energy, each unit i can determine the difference between the global energy of the network when it is off and the global

energy when it is on, given the current states of the other units. This energy gap is simply

$$E_{i \text{ off}} - E_{i \text{ on}} = \Delta E_i = \sum_j s_j w_{ij}$$

If the energy gap is positive, the unit should turn on (or stay on) to minimize the global energy. Otherwise it should turn off (or stay off). In other words, to minimize energy it should behave exactly like a binary threshold unit.

Hopfield originally proposed his model as a theory of memory. Each local minimum corresponds to a stored vector, and the memory is content-addressable because if it is started anywhere near one of the stored states it normally converges on that state. The same kind of network can also be used for perceptual interpretation by defining some of the units as input units and clamping their states on or off to represent the current perceptual input. The other units then settle into a low-energy global state consistent with these boundary conditions. This state represents a locally-optimal solution given this set of inputs, but with no guarantee that it is the global optimum.

The Boltzmann machine architecture^{15,16} is essentially a Hopfield network (with hidden units) that uses a *simulated annealing* search to escape from local minima. This same general idea, with some differences of emphasis, was independently proposed by other research groups^{17,18} at about the same time.

Simulated annealing is a search technique that has been applied to a number of optimization problems.¹⁹ The idea is that we escape from high local minima by adding a random component to the decision process of each unit. In most cases, the unit still takes a step downhill, but occasionally it will take a step uphill instead. More precisely, each unit i computes the ΔE_i value as above, then assumes the 1 state with probability p_i given by the following formula:

$$p_i = \frac{1}{1 + e^{-\Delta E_i / T}}$$

The T term is a scaling factor that controls the amount of noise. It is analogous to a temperature: For large T , p is about .5 and the system assumes states at random, ignoring the constraints in the network; for $T = 0$, the random element is eliminated, and the system behaves as in the pure Hopfield net, moving downhill into the nearest local minimum. At any given

T , once the system has reached thermal equilibrium, the relative probability of being in state A versus state B (see Figure 5) obeys a Boltzmann distribution:

$$\frac{P_A}{P_B} = e^{-(E_A - E_B)/T}$$

Thermal equilibrium does not mean that the system has settled into a particular stable state. It means that the probability distribution over states has settled down, even though the states are still changing. A deck of cards is at thermal equilibrium when it has been shuffled for long enough that the probability of finding any card in any position is $1/52$, even though the cards keep moving around. The best way to reach thermal equilibrium at a given temperature is generally to start at a higher temperature, which makes it easy for the system to cross energy barriers but gives it little preference for the lower energy states. Then we gradually increase the preference for low energy states by reducing T . If T is reduced slowly enough, there is a high probability of ending up in the best global state, or if not there, in a state not much worse than the best. This process of slow cooling is analogous to slow annealing of a metal in order to crystallize it in its lowest energy state.

Boltzmann networks, designed by hand and running on a serial simulator, have been demonstrated successfully on a number of constraint-satisfaction problems, such as the separation of figure and ground in a two-dimensional image. Touretzky²⁰ is investigating ways of implementing symbolic processing, including a production system interpreter and a system that can manipulate lists and tree structures, using Boltzmann-type networks. But most of the interest in Boltzmann machines is due to a learning procedure, described later, that enables them to program constraints into their connections when shown examples of good solutions to a problem.

Hopfield and Tank²¹ investigated a different method to avoid being trapped in local minima: They used real-valued, analog, nonlinear elements that obey the same equation as the units in a Boltzmann machine but send out a deterministic real value instead of a single stochastic bit. Reducing the temperature in a Boltzmann machine is equivalent to raising the gain (the nonlinearity) of their analog units, and they have shown that they can approximate simulated annealing without actually simulating a stochastic system directly.

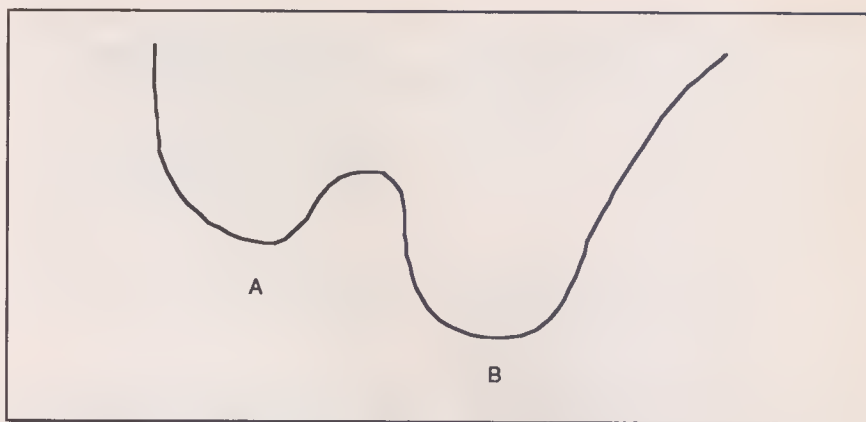


Figure 5. Energy surface with local minimum at A, global minimum at B.

As the gain is increased, single minima of the low-gain energy surface develop a fine structure of smaller minima.

Hopfield and Tank's approximation loses all information about the higher-order statistics. For example, they would represent a stochastic system that spent half its time in the state (1, 1) and half its time in the state (0, 0) by the mean values (0.5, 0.5). But the very same representation would be used for a system that spends half its time in the state (1, 0) and half in the state (0, 1). It is not yet clear how important this loss of higher-order statistical information is.

Networks of this type have been applied to a variety of problems, the most ambitious of which is the Traveling Salesman Problem, for which the network finds moderately good solutions very fast. Hopfield's group has shown a particular enthusiasm for implementing their networks in silicon, rather than simulating them, and a number of small-scale physical implementations have appeared. At the same time, Hopfield has been working with Tank to investigate the possible connection between these networks and real neurons.

Hopfield's group has chosen not to focus on the problem of learning what to do with extra, hidden, units whose required behavior is not specified by the task definition. These are not needed if the experimenter decides how to represent things, but they are crucial if the network is to be given the freedom to develop its own representations. There exists an iterative form of the back-propagation learning procedure that can learn how to use hidden units in a Hopfield and Tank network and can also learn an optimal schedule for varying the gain of the units during

the search process. If the same learning procedure is applied to nets with asymmetric connections, they can be trained to produce sequences.²² The learning procedure is infeasibly slow for large nets, so the research currently divides into studies of search in larger nets, built by hand, and learning in smaller nets.

The Boltzmann machine learning procedure

The Boltzmann learning scheme is surprisingly simple. It has two versions; we will describe the version in which there are input units and output units and the machine must learn to map input vectors into output vectors. We begin with a set of I/O patterns on which the network is to be trained. The goal is to adjust the weights of the network so that, when we clamp the input units into one of these patterns and anneal the network, the corresponding output pattern will appear on the output units. If we clamp the inputs into a pattern that the system has not seen before, we would like the system to generalize according to the underlying regularities in the I/O pairs that it has seen.

The learning cycle has two phases, positive and negative, followed by weight adjustment. During the positive phase, we cycle through the entire set of I/O pairs. Each of these, in turn, is clamped into the input and output units and the rest of the network is then annealed, starting with a high temperature and gradually cooling down to thermal equilibrium at a temperature of 1. Once the system is close to equilibrium, we keep running it for a few more cycles, during which time each con-

nection keeps a record of how often the two units it joins are on at the same time. After all of the I/O pairs have been presented in this way, each connection will have recorded a value, p^+ , which is the fraction of time during the positive phase in which the two connected nodes are on at the same time at thermal equilibrium.

The negative phase is identical, except that only the inputs are clamped; the output units are allowed to settle into whatever states they like. In a network that has learned perfectly, the output units will exhibit the same probability distribution over output vectors as they would if we were still clamping them along with the inputs. During the negative phase we again run for a few more cycles after reaching equilibrium. Each connection records a second value, p^- , the fraction of time in which its two units are both active during the negative phase. If the network is producing all the right answers, the output units will exhibit the same probability distribution whether or not we clamp them and p^+ will be the same as p^- . If p^+ and p^- differ for a given connection, the two probability distributions can be made to match better by changing that connection's weight. If we ignore sampling noise, the difference between p^+ and p^- is exactly equal to the gradient of an information theoretic measure of the difference between the system's output behavior (during the negative phase) and its desired behavior (during the positive phase).¹⁶ So we can perform steepest descent in this difference measure by changing the weight by an amount proportional to $p^+ - p^-$.

Even though Boltzmann machines only contain pairwise connections, the learning procedure allows them to capture higher-order constraints by dedicating hidden units to represent higher-order features. The learning procedure is interesting because it decides what the hidden units should represent, and it typically chooses to use distributed representations.

Unfortunately, the Boltzmann machine learning procedure suffers from all the usual problems of gradient descent in large parameter spaces. On top of this, the estimated gradient is usually inaccurate. The proofs assume that we have a large, unbiased sample of the statistics at equilibrium in the positive and negative phases. In practice, we never quite reach equilibrium, so the statistics are systematically biased. Moreover, unless we are very patient there is also sampling error caused by taking too few samples.

Despite these difficulties, Boltzmann

machines have been used successfully for aspects of speech recognition.²³ The simplicity of the Boltzmann machine architecture makes it feasible to build VLSI chips that contain many units, and it is just possible that this will lead to sufficient speed to make this a practical scheme. Even so, the scaling properties are poor and very large networks will remain impractical unless clever scaling tricks are discovered.

Ten years ago there were no connectionist learning procedures powerful enough to build useful representations with multiple layers of hidden units. Now there are many. The current challenge is to develop faster learning schemes that can be scaled up to networks with millions of modifiable connections. □

Acknowledgments

Scott Fahlman's research was supported by the Defense Advanced Research Projects Agency (DOD), DARPA Order No. 4976, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520. Geoffrey Hinton's research was supported by the Office of Naval Research under contract N00014-86-K-00167 and by the National Science Foundation, grant number IST-8520359. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsors or the US government.

Further reading

Parallel Models of Associative Memory, eds. J. A. Anderson and G. E. Hinton, Erlbaum, Hillsdale, NJ, 1981.

Parallel Distributed Processing: Explorations in the Microstructure of Cognition, eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, 1986.

Cognitive Science 9, special issue on connectionist models and their applications, ed. J. A. Feldman, 1985.

References

- G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed Representation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, 1986.
- S. E. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, MA, 1979.
- S. E. Fahlman, "Design Sketch for a Million-Element NETL Machine," *Proc. Nat'l Conf. AI*, Am. Assoc. for AI, Menlo Park, CA, 1980.
- W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.
- F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, New York, NY, 1962.
- M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.
- D. H. Ballard, G. E. Hinton, and T. J. Sejnowski, "Parallel Visual Computation," *Nature*, Vol. 306, Nov. 1983, pp. 21-26.
- T. J. Sejnowski and C. R. Rosenberg, "NETalk: A Parallel Network that Learns to Read Aloud," tech. report JHU/EECS-86-01, The Johns Hopkins Univ., EE and CS tech. reports, 1986.
- G. E. Hinton, "Learning Distributed Representations of Concepts," *Proc. 8th An. Conf. Cognitive Science Soc.*, Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986.
- L. R. Bahl, F. Jelinek, and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 2, Mar. 1983, pp. 179-190.
- A. G. Barto, "Learning by Statistical Cooperation of Self-Interested Neuron-Like Computing Elements," *Human Neurobiology*, Vol. 4, No. 4, Springer-Verlag, New York, NY, 1985, pp. 229-256.
- J. A. Feldman and D. H. Ballard, "Connectionist Models and their Properties," *Cognitive Science*, Vol. 6, No. 3, Ablex, Norwood, NJ, 1982, pp. 205-254.
- R. A. Hummel and S. W. Zucker, "On the Foundations of Relaxation Labeling Processes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 3, May 1983, pp. 267-287.
- J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat'l Academy of Sciences USA*, Vol. 79, No. 8, Apr. 1982, Nat'l Academy of Sciences, Washington, DC, pp. 2554-2558.
- S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski, "Massively Parallel Architecture for AI: NETL, Thistle, and Boltzmann Machines," *Proc. Nat'l Conf. AI*, Am. Assoc. for AI, Menlo Park, CA, 1983. Dist. by William Kaufman, Inc., Los Altos, CA.
- D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, Vol. 9, No. 1, Ablex, Norwood, NJ, 1985, pp. 147-169.
- S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 6, Nov. 1984, pp. 721-741.
- P. Smolensky, "Schema Selection and Stochastic Inference in Modular Environments," *Proc. Nat'l Conf. AI, AAAI-83*, Am. Assoc. for AI, Menlo Park, CA, 1983, pp. 109-113. Dist. by William Kaufman, Inc., Los Altos, CA.
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4589, Am. Assoc. for the Advancement of Science, Washington, DC, 1983, pp. 671-680.
- D. S. Touretzky and G. E. Hinton, "Symbols Among the Neurons: Details of a Connectionist Inference Architecture," *IJCAI*, Vol. 9, Aug. 1985, Morgan Kaufman, Los Altos, CA, pp. 238-243.
- J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, No. 3, Springer-Verlag, New York, NY, 1985, pp. 141-152.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations

by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Bradford Books, Cambridge, MA, 1986.

23. R. Prager, T. D. Harrison, and F. Fallside, "Boltzmann Machines for Speech Recognition," *Computer Speech and Language*, Vol. 1, No. 1, Academic Press, London, UK, 1986, pp. 1-20.



Scott E. Fahlman is a senior research computer scientist in the computer science department of Carnegie-Mellon University. His primary research interest is in massively parallel computing architecture for such AI problems as recognition, knowledge representation, learning, and simple kinds of search and inference. He has also been active in the definition, implementation, and standardization of the Common Lisp language.

Fahlman is a founder of Expert Technologies Inc. and Lucid Inc. and consults for Siemens Central Research Laboratories in Munich. He received his PhD from MIT in 1977, where he also received his BS and MS.



Geoffrey E. Hinton is an associate professor in the computer science department at Carnegie-Mellon University. His main research interest is in models of computation in the brain.

Hinton received a PhD in AI from Edinburgh University in 1978. He has been a research fellow in AI at Sussex University, a visiting scholar in cognitive science at UC San Diego, and a research scientist at the Applied Psychology Unit in Cambridge, England.

Readers may write to Fahlman at the Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA 15213.

What's new? Ask us!

In the Advanced Technology directorate of ESL (a 2500 person company), our charter is to develop the most imaginative and necessarily unconventional solutions to deal with some of the hardest signal processing problems and missions confronting our Nation.

Our goals:

At ESL, we are only interested in developing Advanced Technology Solutions for missions vital to our National Defense.

Our technology:

From Multi-Gigaflop Advanced Digital Signal Processor Systems, to Full-Custom 1.25 micron CMOS VLSI microcircuits, to Wafer-Scale Integrated Gigaflop Systolic Processors, to MIPS and 68020 Based, Tightly-Coupled Distributed Computing Systems, to High-Speed Data-Flow Systems, to Artificial Intelligence Systems.

Our missions:

From Acoustic Signal Processing, to ESM/ELINT Systems, to Tracking Missions, to Advanced COMINT, to other Tactical & Strategic Surveillance and Reconnaissance.

Our business:

We are currently enjoying record business growth due to the quality of our work and our ability to provide products which meet our customers' needs.

Our requirements:

We are only interested in self-motivated, innovative, responsible, and curious engineering professionals, from the disciplines of Electrical Engineering, to Computer Science/Engineering, to Mathematics, to Physics and other Physical Sciences.

Our needs:

Openings exist at all levels: Managers, Staff Engineers, Engineers, System Architects, and Designers. PhD or equivalent, with clear track record of doing something big.

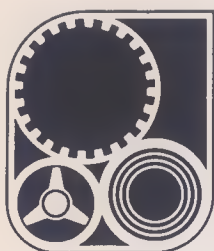
Your move:

At ESL, you'll find excellent salaries and benefits complemented by a stimulating, health-oriented environment. Please send resume to Bijoy G. Chatterjee, Deputy Director, Advanced Technology, Dept. COM-781, ESL, 1345 Crossman Ave., P.O. Box 3510, Sunnyvale, CA 94088-3510 or call him directly at (408) 743-6254. An equal opportunity employer. U.S. CITIZENSHIP IS REQUIRED FOR ALL POSITIONS.

There's a formula for the future.
And it's ESL.

ESL
A Subsidiary of TRW

TRW



9TH Annual International Conference on **SOFTWARE ENGINEERING**

March 30 - April 2, 1987
Monterey, CA, USA

Formalizing and Automating the Software Process

The theme *Formalizing and Automating the Software Process* identifies the two essential activities for realizing the potential of Software Engineering. The first is to rigorously define software creation and evolution processes. The second is to create organizational and automated systems supporting these processes. This theme has guided the definition of a Technical Program organized into Process, Formalism and Automation Tracks, a Tutorial Program providing background information, and a Tools Fair in which a variety of available technology will be demonstrated.

Each of the tracks in the Technical Program includes major talks, papers, workshop reports and panels that collectively expose and integrate relevant, current activities. The Process Track concerns approaches to defining, measuring, assessing, modifying and evolving software processes. Automated and organizational systems supporting these processes are the subject of the Automation Track. The Formalism Track covers both the use of formalism within the processes and the rigorous definition of the processes, their constituent activities and the supporting organizational and automated systems.

The Tutorial Program precedes the other activities and provides in-depth, one-day introductions to a variety of technology pertinent to formalizing and automating the software process. The program covers database technology supporting software engineering, software development environments, knowledge-based programming environments, formal specification and verification techniques, and software reusability.

The Tools Fair includes demonstrations of both production and prototype tools. It also includes in-depth, technical presentations covering particularly interesting and innovative tools. The Tools Fair parallels the Technical Program and its activities are scheduled to allow attendance at both activities.

CHAIR

William E. Riddle
Software Productivity Consortium

PROGRAM CHAIRS

Robert M. Balzer
USC Information Sciences Institute
Kouichi Kishida
Software Research Associates, Inc.

TOOLS FAIR CHAIRS

Larry E. Druffel
Software Engineering Institute
Jack C. Wileden
University of Massachusetts

TUTORIAL CHAIR

Richard E. Fairley
Wang Institute

LOCAL ARRANGEMENTS CHAIR

William M. Murray
General Dynamics Corporation

SPONSORED BY:

The COMPUTER SOCIETY of the IEEE
Technical Committee on Software Engineering

ACM Special Interest Group on Software Engineering

Institute of Electrical and Electronics Engineers, Inc.
Association for Computing Machinery
Agence de l'Informatique

Association Francaise pour la Cybernetique Economique et Technique
The Fellowship of Engineering, United Kingdom
Information Processing Association of Israel

Israel Section of the IEEE
Singapore Computer Society
Singapore National Computer Board
Software Engineers Association of Japan
British Computer Society



THE COMPUTER SOCIETY
OF THE IEEE



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

IEEE

TECHNICAL PROGRAM

Software Processes Are Software Too L. Osterweil

P
R
O
C
E
S
S

Understanding and Assessing the Software Process

W. Curtis, M. Dowson, W. Humphrey

Measuring and Modeling the Software Process

W. Ehrlich, T. Emerson, A. Goel, H. Mills, Y. Miyazaki, N. Murakami, J. Musa

Empirical Studies of the Software Process

participants to be determined

Processes for Achieving Dependable Software

participants to be determined

Some Existing Software Process Models

P. Feiler, G. Kaiser, D. Schwartz, D. Serbanati

Software Process Assessments and Improvements

M. Harandi, J. Kelly, M. Lubars, T. Nomura, D. Ogle, R. Ramnath, K. Schwan, S. Vasudevan

Software Process Management: Lessons Learned From History

B. Boehm, W. Royce and others to be determined

Approaches to Improving Software Processes

V. Basili, L. Druffel, J. Elmore, R. Fairley, K. Kishida, M. Lehman, W. Riddle, D. Rombach and others to be determined

The Use of Formalisms in Software Engineering D. Bjørner

F
O
R
M
A
L
I
S
M

Database Technology and Software Engineering

P. Bernstein, R. Fairley, N. Goldman and others to be determined

Formal Specification Languages

E. Berliner, K. Futatsugi, J. Goguen, B. Kramer, J. Mesequer, K. Okada, P. Zave

More Formal Specification Languages

P. Ladkin, D. Perry, G. Roman

State-of-the-Art in Transformation Technology

W. Feather, W. Scherlis

Practical Application of Specification Technology

participants to be determined

Automating the Software Process V. Stenning

A
U
T
O
M
A
T
I
O
N

Artificial Intelligence and Software Engineering

D. Barstow, B. Boehm, C. Williams and others to be determined

Intelligence in User Interfaces: Is it Necessary?

participants to be determined

Supporting Version Control

D. Perry, J. Winkler

User Interface Support

C. Rathke, S. Reiss

A Sampling of Existing Environments

participants to be determined

Prospects for Future Environments

T. Cheatham, R. Taylor, C. Tully, T. Yokoi and others to be determined

Support for Evolution and Reuse

D. Embley, G. Fischer, S. Katz, A. Lemke, C. Rathke, C. Richter, K. The, S. Woodfield

Individual and International Property Rights: How Serious is the Problem?

R. Baizer, R. Banin, D. Bradin, P. Samuelson and others to be determined

TOOLS FAIR

31 March - 2 April 1987

The Tools Fair runs parallel to the Technical Program, scheduled to allow attendance at both activities. The purpose of the Tools Fair is to provide conference attendees an opportunity to view a variety of production and prototype software development tools, toolsets and environments from both the research community and industry.

Individual tools, integrated toolsets and full-scale environments will be demonstrated in an exhibition area, providing the opportunity to gain first-hand experience and informally discuss technical details. In addition, a sequence of special presentations will be devoted to the description and demonstration of the technical details of particularly interesting or innovative technology.

Applications for space within the exhibition area and proposals for special presentations must be received by 30 January 1987. Presentation proposals will be judged on the depth of technical content, freedom from marketing "hype," and suitability for illustrating interesting and novel aspects. Applicants for space in the exhibition area are advised that marketing activity will not be allowed.

To request exhibition space or propose a special presentation, contact either of the Tools Fair Chairs:

Larry E. Druffel (412) 268 7740
Jack C. Wileden (413) 545 0289

TUTORIALS

30 March 1987

The Tutorial Program includes several in-depth, one-day introductions to a variety of technology pertinent to formalizing and automating the software process:

Database Technology for Software Engineering

John Nestor
Software Engineering Institute
Richard Snodgrass
University of North Carolina, Chapel Hill

Software Development Environments

Peter B. Henderson
State University of New York at Stony Brook
Mark Ardis
Wang Institute

Formal Specification and Verification

Richard A. Kemmerer
University of California, Santa Barbara

Knowledge-based Programming Environments

David R. Barstow
Schlumberger-Doll Research

Software Reusability

John B. Goodenough
Software Engineering Institute

LODGING RESERVATION FORM

CUT-OFF DATE: 23 February 1987

PROCEDURE:

1. Send this form to:
ICSE-9 Housing Bureau
Monterey Peninsula Housing & Convention Bureau
PO Box 1770
Monterey, CA 93940
Complete the charge authorization or enclose a check, for one night's lodging, payable to Monterey Peninsula Visitors and Convention Bureau.
2. Block reservations will not be accepted.
3. Phone reservations will not be accepted by either the bureau or the participating hotels/motels.
4. List a first, second and third choice; your reservation will be made at your most preferred, available hotel/motel.
5. You will receive your room confirmation directly from the hotel/motel.
6. The rates listed do not include room tax.
7. Suite requests must be made in writing to ICSE-9 Suite Request, MPV&CB, PO Box 1770, Monterey, CA 93940.
8. Notify the Monterey Peninsula Visitors and Convention Bureau, in writing, of any cancellations or changes in your reservation before 15 March 1987. After 15 March 1987, contact the confirming hotel/motel directly. Deposits will be refunded only with 72-hour cancellation notice.

(Please type or print)

Name: _____
Organization: _____
Address: _____
Phone: () _____

Room Preference: ☐ Single ☐ Double

Hotel/Motel Preference: please indicate your preference using 1st, 2nd, 3rd, etc. Single: Double:

☐ Casa Munras - standard	\$65	\$65
☐ Casa Munras - delux	\$75	\$75
☐ Doubletree	\$95	\$110
☐ Hilton Inn Resort	\$89	\$99
☐ Hyatt Regency	\$95	\$95
☐ Monterey Beach - gardenside	\$55	\$55
☐ Monterey Beach - oceanfront	\$65	\$65
☐ Monterey Beach - hills view	\$112	\$112
☐ Monterey Beach - canncry row view	\$128	\$128
☐ Montcrey Beach - ocean view	\$144	\$144
☐ Monterey Beach - deluxe ocean view	\$160	\$160
☐ Sheraton	\$95	\$105

Credit Card: ☐ Visa ☐ MasterCard ☐ Choice ☐ AmExpress

Credit Card Number: _____

Expiration Date: _____

Cardholder's Name: _____

Signature: _____

CONFERENCE REGISTRATION FORM

PROCEDURE:

1. Send this form to:
ICSE-9 Registration
Computer Society of the IEEE
1730 Massachusetts Ave., NW
Washington, DC 20036-1903
2. To guarantee your registration, complete the charge authorization or enclose a check payable to Computer Society of the IEEE.
3. Members of sponsoring organizations are eligible for the reduced Member Fees. Please indicate the sponsoring organization and your membership number when claiming these fees.
4. The Conference Fee includes admission to the technical sessions, admission to the Tools Fair, one copy of the Conference Proceedings (except students), refreshments at breaks, and receptions.
5. The Tutorial Registration Fee includes tutorial notes and lunch.
6. Refunds (less a \$15 handling fee) will be made if a request is received in writing by 15 March 1987.

(Please type or print)

Name: _____
Organization: _____
Address: _____
Phone: () _____

TUTORIAL CHOICE (choose at most one):

- ☐ Database (Nestor/Snodgrass)
☐ Environments (Henderson/Ardis)
☐ Formal (Kemmerer)
☐ Knowledge-based (Barstow)
☐ Reusability (Goodenough)

	Advance Registration (Prior to 25 Feb 1987)		
	Member	Non-Member	Student/Retired
Conference	\$160	\$200	\$25
Tutorial	\$125	\$155	\$50
	Late Registration (After 25 Feb 1987)		
	Member	Non-Member	Student/Retired
Conference	\$200	\$250	\$35
Tutorial	\$155	\$195	\$60

Conference Fee: _____
Society: _____
Membership Number: _____
Tutorial Fee: _____
TOTAL FEE: _____

Credit Card: ☐ Visa ☐ MasterCard ☐ Choice ☐ AmExpress

Credit Card Number: _____

Expiration Date: _____

Cardholder's Name: _____

Signature: _____

BOOK REVIEWS

Expert Systems and Fuzzy Systems

Constantine V. Negotia
(Benjamin/Cummings Pub. Co.,
Menlo Park, Calif., 1985, 190 pp.,
\$37.95)

This book provides an excellent introduction to the subject of fuzzy systems and its influence in the design of expert systems. The formal theory of fuzzy sets, knowledge representation in the design of knowledge engineering systems, decision support systems, and management expert systems are the book's major topics.

According to the author, the book's aim is to "give the reader the mathematical background necessary to understand the algorithms used in approximate reasoning." This he has successfully accomplished.

Chapter 1 introduces expert systems and logic programming and gives a brief overview of the rest of the chapters in the book. The logic of inferencing and instantiation and the concept of approximate reasoning in fuzzy systems are elucidated in Chapter 2. Chapter 3 contains a complete and comprehensive introduction to the algebra of fuzzy sets.

In the next three chapters the author considers the role of the mathematics of fuzzy sets in the realization of expert systems. Knowledge representation in

the knowledge databases of an expert system and verbal models form the subject matter of Chapter 4. This chapter provides, through examples, a demonstration of how fuzzy set technology yields realistic norms for manipulating linguistic values. Chapter 5 demonstrates how the application of rules of inference to natural language statements translated into fuzzy sets results in other fuzzy sets; when translated, these sets lead to approximate consequents of the original premises. Moreover, this inferencing mechanism is developed through models of fuzzy systems. The ability to model vagueness using the fuzzy set approach is considered in Chapter 6. The author contends that fuzzy linear programming has now been sufficiently developed to allow the use of verbal models for automatic transfer of knowledge. The process of converting a fuzzy model to a conventional model is also presented in this chapter, enabling the reader to realize that the large repertoire of existing conventional algorithms in the area of expert systems can be easily used in an expert system using the fuzzy set approach.

Chapter 7 concentrates on management expert systems.

Each chapter has a comprehensive

summary section that helps the reader to review material when necessary and a readings section containing not only the relevant references, but also a lucid explanation identifying the scope and pertinent aspects of the subject matter. An appendix on categorial analysis and an extensive bibliography are also included.

Because it provides them with the required formal aspects of fuzzy sets in relation to expert systems, the book is a good reference book for practicing engineers with the relevant background in logic reasoning. It may not be possible, however, to use this book exclusively as a textbook in a course on expert systems, even though the author states so, in the preface.

Actual implementations of expert systems have not been provided; it would have been ideal if a chapter on "real life" systems had been included. The absence of exercises in the book is another disadvantage. However, the overall organization of the book is excellent and it is almost error-free. (The reviewer noticed a typo in the last line of Page 110.)

Pani N. Chakrapani
Associate Professor of Eng./CS
University of Redlands

Fundamentals of Digital Electronics

George Rutkowski and Jerome Oleksy
(Prentice-Hall, Englewood Cliffs, N.J.,
1985, 273 pp., \$24.95)

Targeted to technicians in industry, this book can also be used as a primer for undergraduate students having trouble understanding digital concepts. The authors, in a summary of their goals, have expressed the need to teach digital electronics in an interesting and dynamic way by including easy-to-visualize problems and "many blinking lights" in the laboratory experiments.

The line diagrams are clear and easy to understand, and the chapter summaries at the end of each chapter provide a quick overview.

The philosophy of the authors has been to provide hands-on experience immediately after exposure to the concepts in the text and, in this way, to produce a technician who is not just a textbook expert but also is actually familiar with the hardware.

Some highlights:

Chapter 1 deals with diodes and their applications; Chapter 2, with transistors and their applications; and Chapter 3, field-effect transistors for logic circuits. A brief introduction is given in each chapter. Chapter 3 mentions the construction and behavior of the FET. Practical CMOS inverters and ICs are also discussed, and a quiz is provided at the end of the text to test understanding of the material. Finally, toward the end of the chapter, two experiments are treated in detail.

Other chapters deal with monostables, counters, flip-flops, square-wave generators, gates, and binary numbers. I think it would be unreasonable to expect a chapter on microprocessors that could do justice to the subject, but it is an inescapable fact that microprocessors have become an integral part of digital electronics.

Chapter 16 on digital applications ties together many concepts and circuits introduced in previous chapters. The aim of the authors in this final chapter is to familiarize the student with complete digital systems and with the interaction of several circuits working together to do a task.

I think the authors have succeeded in their goal of exposing the student to a broad range of digital fundamentals and hardware, while preparing them to easily adapt to more extensive systems. However, I did not quite understand how the book encourages team work, one of the authors' stated goals. Synergy is an attitude that cannot, perhaps, be taught so easily.

On the whole, the authors should be congratulated on an important job well done.

Chandan Sen
Research Triangle Institute

"Any clod can have the facts,
but having opinions is an art."

Charles McCabe
San Francisco Chronicle

THE

OPEN CHANNEL

Does more performance mean less utility?

Computer science consists of showing that a task can be done at all. Computer engineering involves doing what you know can be done cheaply. Architectural features that may be useful for doing good computer science can actually get in the way of doing good computer engineering. We make a mistake when we try to use machines built to do computer science to do computer engineering.

Much of computer science is predicated on simply building faster machines. As our computer systems become faster they are also becoming more complex and less deterministic in time. This makes it harder to apply the machines to real problems. The folk wisdom is that the increased speed more than makes up for the increased complexity and decreased determinism. In other words, doing lots of ill-defined, ill-informed things quickly beats doing a few well-defined, well-informed things slowly. I think it's time we examined this piece of folk wisdom.

First, let's look at complexity. We're told that the answer to the complexity problem is tools. But as complexity grows, the tools must necessarily focus more and more on the solution technology and less and less on the problem we're trying to solve. This is an unfortunate trend because the wellspring of both solution quality and efficiency is to be

found in the understanding of the problem domain not the solution domain.

Second, let's consider time indeterminism. Can you imagine building a house with lumber whose dimensions can vary up to 50 percent? In other words, when you order an 8-foot board you might get a 4-foot board and you might get a 12-foot board. Yet order up an instruction on a modern computer and ask your vendor how long it will take to execute. Averages, even if you can compute them, aren't sufficient. If I'm going to connect the computer to reality, I have to know how long this particular instance of the instruction will take. There is a reason why so few bridges are built with flex-straws.

I am skeptical of trading off a few high-quality cycles for lots of low-quality ones. I think the solutions themselves will be less satisfactory and the resulting systems less efficient, and I fear our understanding of the problem domain will not grow. As Richard Hamming has reminded us, "The purpose of computing is insight not numbers." Modern, high-performance computer architectures are making numbers easier but insight harder. This is a "lose."

Scott B. Guthery
Schlumberger Well Services
Austin, Texas

The Open Channel is exactly what the name implies: a forum for the free exchange of technical ideas. Try to hold your contribution to one page maximum in the final magazine format (about 1000 words).

We'll accept anything (short of libel or obscenity) so long as it's submitted by a member of the Computer Society. If it's really bizarre we may require you to get another member to cosponsor your item.

Send everything to Jim Haynes, Computer Center, UC Santa Cruz, CA 95065.
Bitnet: haynes@ucsc; Arpa: ucsc!haynes@ucbvax.berkeley.edu.

New books from ...



COMPUTER
SCIENCE
PRESS

**DIGITAL TRANSMISSION
SYSTEMS AND NETWORKS,**
Volume I: Principles, and
Volume II: Applications,
*Michael J. Miller and
Syed V. Ahamed*

This two-volume work to be published in early 1987 emphasizes engineering aspects of integrated communications systems for telephony, television, data, and other services. Available Spring 1987; Volume I: 0-88175-094-8, \$50.00; Volume II, 0-88175-176-6, \$50.00.

**LOCAL AREA AND MULTIPLE
ACCESS NETWORKS,**
Raymond Pickholtz, Editor

This outstanding volume covers the subject of Local Area Networks (LAN). Topics by well-known experts in the field include efficiency, congestion control, performance evaluation, routing schemes, flow control, and more. ©1986, 0-88175-143-X, 291 pages, \$50.00.

**CURRENT ADVANCES IN
DISTRIBUTED COMPUTING AND
COMMUNICATIONS,**
Yechiam Yemini, Editor

Recognized experts on state of the art research have contributed chapters to this excellent text. The book is divided into two sections: the first covers research on performance analysis of distributed systems; the second deals with distributed processes and their communications, parallel systems, and VLSI systems. ©1987, 0-88175-128-6, 377 pages, \$49.95.

**Z80 ASSEMBLY LANGUAGE
PROGRAMMING**

Based on the AMS Tutorial Program
Ivan Tomek and Peter W. Steele

This book, and the diskette available for it, is an excellent tool for those who want to learn assembly language programming on an 8-bit computer that uses the Z80 processor and the CP/M operating system. Available Spring 1987, 0-88175-146-4, price not yet set.

Complimentary examination copies of college texts are available with a written request on school letterhead. Please identify course name and number, estimated course enrollment, and current text being used.

Orders from individuals must be prepaid. Prices are subject to change without notice. Postage and handling is \$2.50 for the first book, \$1.50 for each additional title. State and local taxes apply.



COMPUTER SCIENCE PRESS, INC.
1803 Research Blvd., Dept. C187
Rockville, MD 20850
(301) 251-9050

UPDATE

News from the IEEE-CS Committee on Public Policy

New law protects privacy of electronic communications

Jerry P. Jennings, Chair, Subcommittee on Security and Privacy

The Electronic Communications Privacy Act, which is intended to prevent eavesdropping on electronic communications, was signed into law on October 21, 1986. The act will protect the privacy of electronic communications, including computer-to-computer communications, electronic mail, remote computing services, and even cellular car phones. Up to \$10,000 fines can be imposed.

The content and progress of this legislation has been an active area of concern to the Security and Privacy Subcommittee of the IEEE Computer Society's Committee on Public Policy. The subcommittee places special emphasis on impact studies and US legislative activity and develops positions regarding issues of significance to the Computer Society and its members.

The Electronic Communications Privacy Act was introduced by Senators Charles Mathias (R-Md) and Patrick Leahy (D-Vt) as Senate Bill S.1167. The House version, H.R. 4952, was authored by Rep. Robert Kastenmeier (D-Wis). The bill was passed by voice vote in the Senate on October 1, by the House the next day, and was signed into law three weeks later. Copies of the bill may be obtained by writing to Heidi James, IEEE Washington Office, 1111 19th St., NW, Suite 608, Washington, DC 20036.

Lynn W. Ellis, chairman of the IEEE TAB/USAB Committee on Communications and Information Policy, testified regarding the proposed legislation. He urged, in particular, that the technologically restrictive wording of the bill be changed so as not to interfere with the rapid deployment of new technology when it is invented. The issues on which he testified were resolved and incorporated in the final bill.

The bill was endorsed by most of the providers of electronic mail and associations such as the E-Mail Association, GTE-TelNet, and ADAPSO. It was also strongly endorsed by the American Civil Liberties Union, which recommended that efforts be made to ensure the protection of new electronic communications without stifling technical or social innovation or inhibiting the free flow of

information. The ACLU is mostly interested in the protection of data and other forms of nonpublic electronic communications while in transit or held by a third party and any backup involved in the transmission.

The Justice Department, on the other hand, took issue with the proposed legislation on grounds that it would hamper law enforcement in a serious way. A Justice Department spokesman stated that Title III, the Omnibus Crime Control Act, works well and provides an appropriate balance between citizens' rights to privacy and law enforcement and societal interests in preventing crime and apprehending criminals. But he also admitted that Title III should be revalidated periodically for required revisions as technology advances.

This author believes that a well-written law was needed to punish wrong-doers who access private communications for illegal gain. But the law would only apply after the fact and after the damage has been done. Existing laws to prevent the use of information obtained in illegal wiretaps, or by tampering with the US mail, do apply after the fact—when proof of the illegal use of the private information so obtained is available.

Any person or enterprise that wishes to protect its private communications can do so electronically right now. An enterprise protects its paper communications by keeping them under lock and key and its verbal dealings by keeping them behind closed doors. It could protect its telephone conversations through scrambling and its digital information through encryption. (Members interested in encipherment/decipherment devices for personal computers are urged to refer to the April 1985 issue of *Infosystems*, the magazine for information systems management, published by the Hitchcock Publishing Company, Wheaton, IL 60188.)

The COPP Subcommittee on Security and Privacy will now be working on further position papers for endorsement by the Computer Society's Board of Governors. If you are interested in joining the subcommittee, please contact the COPP secretary or the subcommittee

chair at the following addresses: Paul Davis, 41 W. Huron St., Jackson, OH 45640; Jerry P. Jennings, Laddis Corp., PO Box 2008, Dearborn, MI 48123.

Dictionary project publishes glossary

The first of a series of glossaries of computing terminology has been approved and published by the IEEE Standards Board. The standard, ANSI/IEEE Std 1084-1986, *IEEE Standard Glossary of Mathematics of Computing Terminology*, may be purchased for \$8 US, plus handling, from three sources: IEEE Computer Society Press, Worldway Postal Center, Los Angeles, CA 90080, (order number 970; phone (800) CS-BOOKS or (714) 821-8380); IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854 (phone (201) 981-0060); or IEEE Computer Society, 2 Avenue de la Tanche, B1160 Brussels, Belgium (phone 32.2660.11.43).

This is one of a series of incremental dictionary standards that will encompass the entire computer field.

P1085, *Computer Applications Terminology*, is completing balloting and is scheduled for Standards Board approval this month.

P1088, *Computing Environment Terminology*, and P1086, *Theory of Computing Terminology*, are scheduled for ballot in the spring.

P1087, *Computer Hardware Terminology*, and P1089, *Computing Methodologies Terminology*, are scheduled for ballot in the fall.

The complete P610 standard, *Computer Dictionary*, includes terms from all the above and is scheduled for ballot in the spring of 1988, together with the revision of ANSI/IEEE 729-1983, *IEEE Standard Glossary of Software Engineering Terminology*.

The sponsor of this effort is the IEEE Computer Society Standards Coordinating Committee. Volunteers are welcome and should contact Jane Radatz, Logicon, Inc., PO Box 85158, San Diego, CA 92138; (619) 455-1330.

ACM and IEEE-CS launch Fall Joint Computer Conference

Two professional computing societies, the Association for Computing Machinery and the IEEE Computer Society, have now combined their national fall conferences into one joint meeting—the Fall Joint Computer Conference. It replaces the ACM National Conference and the Computer Society's Comcon Fall.

FJCC took place November 2-6 at the Dallas, Texas, Infomart, which will also be the site for the second meeting in the fall of 1987. More than 3000 people attended all or part of the five-day 1986 event, which featured two days of tutorials, three days of technical sessions, and exhibits.

On Sunday and Monday, November 2 and 3, tutorials were offered in six areas: artificial intelligence, computer design, modeling and measurement, computer engineering, programming systems, and software engineering.

The three-day technical program, which began the next day, featured 74 sessions in nine arenas: education, software systems, artificial intelligence, supercomputing, algorithms, modeling and measurement, computer design, international development, and operating systems and databases.



RECEPTION. Fall Joint Computer Conference registrants previewed the exhibit area and the Dallas Infomart facility Monday evening, November 3. Among those attending the Conference and Exhibitors Reception were (above, left to right) Stanley Winkler, conference chair; Ted Landberg, member of the Public Relations Media Committee; and J.T. Cain, acting president of the Computer Society. ACM and the Computer Society cosponsored FJCC, which Winkler termed "the best computer conference of the decade." His personal criterion for the best conference—the one at which he's having the most fun—was met, Winkler said, because he "picked people who made it happen." Those people included Harold Stone, who was responsible for the technical sessions, and Toni Shetler, who chaired the Professional Education Program.

The breadth and depth of the FJCC program, according to Conference Chair Stanley Winkler, probably exceeded that of any conference held in the last decade. He did, however, note that the resulting 1248-page conference proceedings made "heavy reading."



KEYNOTERS. H. Ross Perot (above left), an entrepreneur who founded Dallas-based Electronic Data Systems with \$1000 after quitting his job as an IBM salesman in 1962, gave the FJCC industrial keynote address Tuesday, November 4. Perot sold EDS to General Motors two years ago but continued as chairman of the computer services company until shortly after FJCC. (The GM Board of Directors' decision to sever ties with Perot and buy back his stock was announced December 1.) Wednesday's plenary session featured two keynote speakers: C. Gordon Bell (above right), assistant director for the Computer and Information Science and Engineering Directorate of the National Science Foundation, and Kenneth G. Wilson (not shown), director of Cornell University's supercomputer center.



TUTORIALS. The FJCC Professional Education Program consisted of 20 preconference tutorials held Sunday and Monday, November 2-3. Above, John D. McGregor of Murray State University coaches Bob Hagen of EDS toward completion of a hands-on exercise using techniques just described in "Building Expert Systems." McGregor's tutorial was one of nine on artificial intelligence.

ACM holds contests, presents awards during FJCC

ACM conducted two computer chess tournaments and a student programming contest during the Fall Joint Computer Conference, cosponsored with the IEEE Computer Society and held November 2-6 in Dallas, Texas. The association also presented five major awards at the Thursday, November 6, plenary session.

Awards. John Hopcroft of Cornell University and Robert Tarjan of Princeton University received ACM's most prestigious award—the Turing Award—for their individual and joint contributions in algorithms. The award is named after A.M. Turing, an Englishman who created some of the mathematical foundations of computer science in the 1930's, and is accompanied by a \$2000 prize.

The ACM Distinguished Service Award was presented posthumously to Clair Maple for "a lifetime of innovative contributions and distinguished leadership in the use of computers in higher education." Maple served for many years as director of the Computer Center at Iowa State University.

Herbert Maisel of Georgetown University received ACM's Outstanding Contribution Award. Maisel was cited in particular for his service as chair of the Inter-society Liaison Committee.

Donald E. Knuth of Stanford University received the Software System Award for the "design and implementation of TEX, an innovative tool for the computer composition of documents of high typographical quality."

The Grace Murray Hopper Award, which honors the outstanding young computer professional of the year, was presented to William N. Joy of Sun Microsystems for his work on the Berkeley Unix operating system. The award recipient must have been 30 years of age or less at the time of the qualifying contribution. A prize of \$1000 is supplied by the Sperry Corporation.

Chess. The FJCC hosted two computer chess tournaments and crowned two champions. Belle, a program developed at Bell Laboratories by Ken Thompson and Joe Condon, won the ACM's 17th North American Computer Chess Championship. Mephisto, a program developed for Hegener and Glaser AG by Richard Lang, won the Sixth World Microcomputer Chess Championship.

Belle won all five of its games to recapture a title that it had won several times in the early 1980's. The program ran on a PDP 11/23 and searched chess trees with as many as 30,000,000 positions per move. Finishing in second place

was Lachex, which ran on a Cray XMP and was developed by Burt Wendroff at Los Alamos National Laboratory.

In the microcomputer championship, 14 programs competed for seven rounds. Mephisto won all but one of its games. In second place was Fidelity International's product, Fidelity 2533, which defeated a grand master in tournament play earlier in the year.

Programming. Forty-two teams of student programmers from 33 colleges and universities throughout Louisiana, Arkansas, Oklahoma, and Texas competed in the 17th Annual ACM South Central Region Scholastic Programming Contest. The winners, teams from the University of Texas at San Antonio and from Rice University, will go on to the international competition held during the ACM Computer Science Conference, February 17-19 in St. Louis.

The contest challenged teams to solve five programming problems in a four and one-half hour period. Teams were then ranked on the basis of number of problems solved, time of completion, and number of tries.

According to the contest director, Tom Nute of Texas Christian University,



STATUS REPORT. ACM President Paul Abrahams delivered his state-of-the-ACM address during the Thursday, November 6, FJCC plenary session. He cited the Special Interest Groups as one of the association's strengths and outlined plans for budget reorganization and membership growth.

teams typically go to great lengths to develop strategies to deal not only with the problems but with such questions as "Who will work on which problems, when do you give up and move on to another problem, and how do you take advantage of contest procedure?"



PROGRAMMING CONTEST. "We're winning," Thane Mitchell replied when asked—several hours before the competition ended—how his team was faring in ACM's South Central Region Scholastic Programming Contest. Confidence and team strategy paid off. Thane (watching from the background in the photo above) and teammates (foreground, left to right) Ian Mitchell, Bruce Tomlin, and Richard Lial did, indeed, win. The University of Texas at San Antonio team completed four of five problems in a total time of six hours and 18 minutes, bettering the second-place Rice University team by almost two hours. Part of their winning strategy—an important element according to the contest director, Tom Nute—was to allow only the two fastest typists on the terminal and to have one of them typing at all times.

AFIPS seeks nominees for three 1987 awards

Nominations are open for three awards to be presented by the American Federation of Information Processing Societies—the Harry Goode Memorial Award, the Education Award, and the AFIPS-Fortune Magazine Product-of-the-Year Awards.

The Harry Goode Memorial Award is given each year in memory of Harry H. Goode, a pioneer and leader in the field of systems engineering. It honors either a single contribution of outstanding significance in theory, design, or techniques in information processing, or an accumulation of contributions that represent an outstanding contribution. The award provides a medal, a certificate, and a \$2000 honorarium.

The Education Award honors contributions through publications, innovative teaching, curriculum development, and/or leadership. It consists of a certificate and a \$1000 honorarium.

The Product-of-the-Year Awards, established in 1985, recognize outstanding products in the computer field. One award and up to two meritorious mentions may be given in each of three categories: hardware, software, and sys-

tems. Nominated products must be innovative, have a potentially significant impact on the computer industry, and be commercially available within the United States.

To submit a nomination, request and complete the appropriate form and send it to Stanley Winkler, c/o AFIPS, 1899 Preston White Dr., Reston, VA 22091. Nominations must arrive by February 1, 1987, except for product nominations, which are due March 15, 1987.

Nominations open for Eckert-Mauchly Award

Nominations are now being solicited for the Eckert-Mauchly Award. The joint ACM-IEEE Computer Society award is presented annually to a person or persons for technical contributions to computer and digital systems architecture.

The 1987 award, which carries an honorarium of \$1000, will be presented at the 14th International Symposium on Computer Architecture, June 3-6, 1987, in Pittsburgh.

Center offers supercomputer time

A new supercomputer facility has been completed in Toronto, Canada. The facility houses a Cyber 205 supercomputer, and the machine is underutilized. To help offset operating costs, MEM Marketing Services is offering computer time to outside scientists and engineers. Researchers who have a need for supercomputer time may call Mike Madryga at (416) 475-7097 to discuss their needs.

Each nomination should describe in 200-500 words the candidate's contributions and be accompanied, where possible, by the names, addresses, and telephone numbers of others qualified to supply further information about the candidate.

Nominations should be submitted no later than March 15, 1987, to John Riganati, Supercomputing Research Center, 4380 Forbes Blvd., Lanham, MD 20706.



BUS
THE RACE
IS ON!

BUSCON/87-West

The Bus/Board Users Show & Conference

January 19-21, 1987

Los Angeles Airport Hilton Hotel
Los Angeles, California

DISCOVER:

- Which vendors are winning the race
- What products are available
- New ways to solve your technical needs
- How to keep you and your company up-to-date in this rapidly changing field
- All about new—and special—Bus architectures

SEE: Hands-on exhibits with new Bus/Board products

LEARN: Through tutorial seminars and technical sessions on Emerging Bus Technologies and Features, plus new applications ideas for your current projects

For more information, write Bill Weber
BUSCON, 17100 Norwalk Blvd., Ste. 116,
Cerritos, CA 90701-2750
or call Bill at (213) 402-1610

Join the Computer Society Today

Schedule of Fees

Membership dues and publication subscriptions are annualized to December 31. Depending upon the date your application is received by the Computer Society, pay the full annual amount or one-half the annual amount as indicated below. Membership expires December 31.

	Half-Year Mar 1-Aug 31	Full-Year Sept 1-Feb 28
1 I don't belong to the IEEE and I want to join just the Computer Society	☐ \$19.50	☐ \$39.00
2 I already belong to the IEEE and I want to join the Computer Society IEEE Member Number _____	☐ \$ 7.50	☐ \$15.00
3 I don't belong to the IEEE and I want to join both the Computer Society and the IEEE* Full IEEE members receive <i>Spectrum</i> , <i>The Institute</i> , and many additional benefits. (Total amount to be paid includes annual dues and regional assessment, if any.)		
I reside in Region 1-6 (United States)	☐ \$41.00	☐ \$82.00
I reside in Region 7 (Canada)	☐ \$38.50	☐ \$77.00
I reside in Region 8 (Europe, Africa, or the Middle East)	☐ \$37.00	☐ \$74.00
I reside in Region 9 (Latin America)	☐ \$33.50	☐ \$67.00
I reside in Region 10 (Asia and Pacific)	☐ \$34.00	☐ \$68.00

*ACM members who join both IEEE and the Computer Society are entitled to a deduction of \$5 off the full-year rate, \$2.50 off the half-year rate.

4 In addition to my automatic subscription to *COMPUTER*, I would also like to subscribe to:

	(issues per year)		
<i>IEEE Expert</i> (3151)	4	☐ \$ 6.00	☐ \$12.00
<i>IEEE Computer Graphics and Applications</i> (3061)	12	☐ \$13.00	☐ \$26.00
<i>IEEE Design and Test</i> (3111)	6	☐ \$ 8.50	☐ \$17.00
<i>IEEE Micro</i> (3071)	6	☐ \$ 8.50	☐ \$17.00
<i>IEEE Software</i> (3121)	6	☐ \$ 8.00	☐ \$16.00
<i>Transactions on Computers</i> (1161)	12	☐ \$ 9.00	☐ \$18.00
<i>Transactions on Pattern Analysis and Machine Intelligence</i> (1351)	6	☐ \$ 7.50	☐ \$15.00
<i>Transactions on Software Engineering</i> (1171)	12	☐ \$ 9.00	☐ \$18.00
<i>Journal on Lightwave Technology</i> (4301)	12	☐ \$ 9.00	☐ \$18.00
<i>Journal of Robotics and Automation</i> (4401)	4	☐ \$ 5.00	☐ \$10.00
<i>Journal of Solid State Circuits</i> (4101)	4	☐ \$ 4.00	☐ \$ 8.00
<i>Proceedings of the IEEE</i> (5011) (available only to IEEE members)	12	☐ \$ 8.50	☐ \$17.00

☐ Diners Card ☐ VISA ☐ MasterCard ☐ American Express

Mo	Yr

Mo	Yr

☐ Check or money order enclosed.
Make payable to IEEE.

\$ _____
Total amount remitted with this application

Check box for additional information:

☐ Student Membership Application

☐ Publications Catalog

☐ Bulk air option (two-to-six-week delivery of publications) for members in Western Europe, Latin America, and the Far East

I hereby make application for IEEE Computer Society membership and, if elected, will be governed by IEEE's and the Society's Constitutions, Bylaws, and Statements of Policies and Procedures.

MAILING ADDRESS

First Name		Middle initial(s)	Last name	
Street address		City	State/Country	Zip

EDUCATION (highest level completed)

Course	Degrees received	Date
Name of educational institution	Date of birth	☐ Male ☐ Female

OCCUPATION

Title or position
Firm name
Firm address
City
State/Country
Zip
Affiliate society membership no. (if applicable)
Grade

REFERENCE (an IEEE member; if unknown, a managerial person who knows you professionally)

Name (print in full)	IEEE Member No. (if applicable)
Position	
Street address	
City	
State/Country	
Zip	

MAIL TO: IEEE COMPUTER SOCIETY, 10662 LOS VAQUEROS CIRCLE, LOS ALAMITOS, CA 90720-2578

NEW PRODUCTS

NEC offers laptop

NEC Home Electronics has announced an IBM PC-compatible portable PC called the Multispeed. The NEC laptop operates at both 4.77 MHz and 9.54 MHz. It uses the MS-DOS 3.2 operating system, but is also compatible with earlier versions.

The Multispeed features NEC's V-30 proprietary microprocessor chip, two 3.5-inch floppy disk drives, a detachable screen with an 80-character-by-25-line display, 640K bytes of RAM, 512K bytes of ROM, built-in firmware with windowing, and a separate numeric keypad. The portable contains parallel, serial 232-C, and RGB video interface ports, plus an external floppy disk controller port.

Options include a 300/1200-baud Hayes-compatible modem, a PC data transfer package, and a carrying case.

The Multispeed measures 13.6 by 12 by 3 inches and weighs under 12 pounds. It costs \$1995. For more information, contact NEC Home Electronics, Computer Products Div., 1255 Michael Dr., Wood Dale, IL 60191-1094; (312) 860-9500.

Reader Service Number 40

Datavue offers laptop

Datavue has announced the Spark, an IBM PC-compatible laptop computer. The nine-pound laptop comes with a 3.5-inch floppy disk drive, 384K bytes of DRAM, a birefringent screen, and serial, parallel, RGB, and composite ports. The 80C88 processor runs at either 4.77 or 9.54 MHz.

Options include dual 3.5-inch floppy drives, an internal 300/1200-baud modem with two RJ11 jacks, and EL backlit screen, and a 256K-byte memory board.

The Spark costs \$995. Contact Datavue, Portable Computer Div., One Meca Way, Norcross, GA 30093-2919; (404) 564-5668.

Reader Service Number 41

PC-DOS programming languages support 80386

Intel has announced the C 386 Compiler and RLL 386 Relocation, Linkage, and Library Tools package 80386 development tools for IBM PC-AT and AT-compatible PCs running DOS 3.0 or later. The DOS tools can be used on stand-alone PCs, or on PCs linked to other PCs or to VAX and Micro-VAX computers through the Intel OpenNET network.

The C 386 Compiler produces object code compatible with Intel's other 80386 languages, allowing modules written in different languages on different hosts to be linked using the RLL 386 linkage utilities.

The RLL 386 Relocation, Linkage, and Li-

Micro CAD/CAM has CAD for the Macintosh

Micro CAD/CAM offers a computer-aided design software program for the Apple Macintosh called MGMS Professional CAD for Mac. (MGMS stands for Micro Graphic Manufacturing Station.)

The program offers cartesian, absolute, incremental, and polar measuring systems, and English or Metric dimensioning accurate to four decimal places. (Calculations are made to 15 decimal places.) It stores graphics in PICT format and can transfer graphics to and from Macintosh programs. An optional module allows graphics to be transferred to and from personal and mainframe computers using IGES. For example, the module can send Macdraw graphics to an IBM PC, converting them into AutoCAD graphics.

MGMS Professional CAD for Mac creates objects by grouping lines, arcs, and circles.

Objects can be translated, rotated, or mirrored. Up to 64 layers of component parts can be shown or hidden, grouped, or manipulated separately.

Other features include a geometrical module, splining, scrolling, and movable axes. A library stores up to 50 symbols and drawings. The number of libraries is limited by disk space available.

Output is supported by the Apple ImageWriter I and II, Apple Laserwriter, and various size A through E pen plotters.

MGMS Professional CAD for Mac retails for \$799. For more information, contact Micro CAD/CAM, 3230 Overland Ave., #105, Los Angeles, CA 90034; (213) 376-6860.

Reader Service Number 43

New HP products expand HP Touchscreen

Hewlett-Packard has announced the HP expanded-memory board and the Microsoft Windows environment manager for the HP Touchscreen PC. The company also announced for Touchscreen new versions of MS-DOS 3.2, Lotus 1-2-3, and upgrades for HP Execudesk, Memomaker, and Personal Card File.

The HP expanded-memory board comes with 512K bytes of memory that can be increased in 256K-byte increments to 2M bytes per board. As many as four boards can be used simultaneously. The board costs \$695 for 512K bytes, \$180 for each 256K-byte increment, and \$595 for the numeric coprocessor.

The board also comes with a RAM disk drive that reputedly speeds the performance of applications software, a Save utility that allows users to copy RAM disk to a micro-floppy or hard disk, the HP human interface link to connect input devices, and a printer

buffer that allows one file to be edited while another prints.

Microsoft Windows for Touchscreen costs \$175. Lotus 1-2-3 costs \$495; Execudesk, \$95; MS-DOS 3.2, \$85; the Memomaker upgrade, \$85; and the Personal Card File upgrade, \$135.

For more information, write Inquiries Manager, Hewlett-Packard, 1820 Embarcadero Rd., Palo Alto, CA 94303, or call (800) 367-4772 for the number of the nearest sales office.

Reader Service Number 44

C compiler applies AI

The C86Plus from Computer Innovations is based on technology that applies artificial intelligence techniques to produce optimized code, according to the company. This technology also reputedly allows their C compiler to take advantage of hardware architectures such as Intel's 80286 and 80386 microprocessors.

Computer Innovations claims a 70 percent increase in speed of the C86Plus running the sieve benchmark over the current C86 version 2.3, the company's prior C compiler.

The C86Plus includes ANSI C library functions such as register variables, structure assignment, function prototypes, type modifiers, long double 80-bit floating-point, and enumerator data types.

The C86Plus retails for \$497. Contact Computer Innovations, 980 Shrewsbury Ave., Tinton Falls, NJ 07724; (201) 542-5920.

Reader Service Number 45

Kaypro introduces 386-based supermicros

Kaypro has announced three super-microcomputer systems based on the Intel 80386 microprocessor.

The Kaypro 386, Model A entry-level unit includes a 1.2M-byte floppy disk drive with 512K bytes of RAM and a floppy disk and hard disk controller board. The Model E includes a 1.2M-byte floppy disk drive, a 40M-byte hard disk, and 1M byte of RAM. The Model-N Network File Server comes with a 170, 280, or 380M-byte ESDI hard disk drive. All models are expandable up to 660M bytes of hard disk storage with 16M bytes of RAM. They are available with a

monochrome, CGA, or EGA video display. Kaypro will also offer two optional 5M-byte per minute, 60M-byte cartridge tape units.

The Model A costs \$4995. The Model E with 40M bytes costs \$6295; with 130M bytes, \$8595. The Model N costs \$11,450 with 170M bytes; \$14,950 with 280M bytes; and \$19,950 with 380M bytes.

For more information, contact Kaypro, PO Box N, Del Mar, CA 92014; (619) 481-4300.

Reader Service Number 46

New Micro expands product line

New Micro has announced an in-circuit emulator for the Intel 80386, a symbolic debugger for MICE in-circuit emulators, an in-circuit emulator for Motorola 68000 and 68010 microprocessors, and a development system for Intel 8086 and 8088 and NEC V20 and V30 microprocessors.

The MICE-32/80386 in-circuit emulator for the Intel 80386 can run with an IBM PC-XT, PC-AT, or compatible, or with a VAX or MicroVAX using VMS or Ultrix. The emulator can also operate as a stand-alone system with any terminal device. It features 32-bit-wide registers and a 104-bit-wide trace buffer. It costs \$15,000.

The MSD-3 language-structured symbolic

debugger supports New Micro's MICE in-circuit emulators. It costs \$840 running on a PC, \$2000 running on a VAX with VMS or Ultrix, and \$1650 running on a MicroVAX with VMS or Ultrix.

The MICE-16/68000 is a 16-bit in-circuit emulator for the Motorola 68000 and 68010 microprocessors. It costs \$9000.

The MICE-16/8086F is a 16-bit in-circuit emulator for the Intel 8086 and 8088 and NEC V20 and V30 microprocessors. It costs \$9000.

For more information, contact New Micro, 16901 S. Western Ave., Gardena, CA 90247; (213) 538-5369.

Reader Service Number 47

5030 emulation for Domain

Apollo Computer has announced the Domain 5080 Emulator, an IBM 5080 graphics terminal emulation that lets users simultaneously run both mainframe and workstation applications from the same workstation display. The product combines hardware and software and is compatible with Apollo's family of Domain workstations.

The Domain 5080 emulates IBM 5080 graphics terminals and the standard 5080-style peripherals, including keyboard, lighted-function program keyboard, data tablet, and input dials. The emulator uses graphics software to emulate 5080 graphics order codes and display-list functionality, and it supports Apollo's graphics service routines.

The Domain 5080 Emulator is available in a bundled system that includes Spectra-graphics DSCC, interface boards, Domain DSP90 gateway server, and software site licenses for all workstations on a network. Prices for local configurations range from \$44,500 for 4 users to \$54,500 for 16 users (or \$3400 per user for a 16-user system). Remote configurations range from \$59,900 for 4 users to \$69,900 for 16 users. Upgrades for existing DSCCs are available.

For more information, contact Apollo Computer, 330 Billerica Rd., Chelmsford, MA 01824; (617) 256-6600.

Reader Service Number 48

DISTRIBUTED INFORMATION SYSTEMS: NETWORKS AND APPLICATIONS

a Workshop Sponsored by the
IEEE Information Theory, Computer and Communications
Washington D.C./Northern Virginia Chapters
On Monday, February 23, 1987 8:00 - 6:00



- Distributed Algorithms
Prof. Robert Gallager, Massachusetts Institute of Technology
- Protocols for Key Management and Authentication
Prof. Raymond Pickholtz, George Washington University
- Information Infrastructures
Dr. Vinton Cerf, Corporation for National Research Initiatives

- Recent Developments in the ISO Encryption Standard
Dr. Dennis Branstad, National Bureau of Standards
- Networking with Radio
Prof. Anthony Ephermides, University of Maryland
- Interoperability and Distributed Systems: A Database Perspective
Mr. Steve Oxman, OXKO Corporation

Mail check along with completed registration form to:

IEEE INFORMATION THEORY GROUP - WORKSHOP
12498 Fan Leaf Court
Fairfax, Virginia 22033

Registration confirmation, map and directions will be sent to those who register before February 9, 1987. Registration after February 9, 1987 will be at the door and will cost an additional \$10.00.

REGISTRATION FORM:

Yes! Please register me in the Workshop on Distributed Information Systems to be held on February 23, 1987 in Washington D.C.. Enclosed is a check for (please check one box)

	Preregistration Prior to Feb. 9, 1987	Membership Number
IEEE Member	\$60.00 ☐	_____
IEEE Student Member	\$45.00 ☐	_____
Nonmembers	\$75.00 ☐	_____

Registration fee includes coffee and donuts, a light lunch and afternoon refreshments.

For further information contact Dave McDysan (703) 734-2263, or Dr. Bijan Jabbari (301) 428-5660.

Workshop will be held at the George Washington University, Washington D.C. Marvin Center Continental Ballroom on the corner of 21st and H Streets.

made payable to: IEEE Information Theory Group.

NAME _____

AFFILIATION _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

TELEPHONE # _____

Tops Print links IBM PCs to Apple Laserwriters

Centram Systems West has announced Tops Print, a software program that enables IBM PCs and compatibles to print directly to shared Apple Laserwriters on an Appletalk network.

According to the company, the program captures the character stream resulting from a PC user issuing a standard print command from within an application and translates the characters into Postscript, the page description language used by the Laserwriter. The file is then automatically sent to the first available Laserwriter on the network and printed. A spooler feature lets the user continue working while waiting for the print job to finish.

The software can also print to an Appletalk Imagewriter II, a local printer, or a Postscript-driven typesetter, such as the Linotronic.

Tops Print costs \$189 per PC. For more information, contact Centram, 2650 Ninth St., Berkeley, CA 94710; (415) 549-5900.

Reader Service Number 49

Expert system for Wang VS

Coyne Kalajian has announced ExpertR, an expert system shell for the Wang VS. ExpertR allows users to develop rule-based interactive expert systems by selecting from menus and filling in the blanks to create expert programs. It permits integration of images and text, so that the expert system can ask the user questions that require a visual image to understand.

ExpertR retails for \$25,000. For more information, contact Coyne Kalajian, 1701 N. Fort Myer Dr., Arlington, VA 22209; (703) 522-7026.

Reader Service Number 50

Imaging for the lab

Microtex has announced the Microtex 8700 Scientific Imaging System for the laboratory environment. The product includes digital imaging acquisition, processing, and analysis techniques to permit exploration of experimental regimes. It has 12 bits per pixel resolution.

The systems feature a turnkey workstation that includes imaging hardware and image rendering software. Each Microtex 8700 includes a MicroVAX II computer. Users may share data and resources with other DEC and IBM computers using Ethernet links.

Prices range from about \$110,000 to \$130,000 depending on configuration. For more information, contact Microtex, 80 Trowbridge St., Cambridge, MA 02138; (617) 491-2874.

Reader Service Number 51

3-D CAD/CAM system costs around \$15,000

Micro Engineering Solutions has announced Solution 3000, an IBM PC-based, three-dimensional computer-aided design and manufacturing system. A complete turnkey system, including hardware, software, training, and warranty, costs under \$15,000. Software can be purchased separately for approved configurations.

Solution 3000 targets design and manufacturing engineers who need features such as product design and surface modeling; fixture and gage design; die design; wood model prototype production, numerical control machining; coordinate measuring; and detailing and editing.

The wire-frame surfaced modeling system allows users to observe their models with up to four different views simultaneously.

Geometry added or edited in any viewport will be reflected in all of them.

On-screen icons, pop-up menus, and help screens walk first-time users through the process, while more experienced users can employ programmable tablet menus, programmable keys and icons, and user-defined macros.

Bidirectional interfaces permit communication with other CAD/CAM systems. Available formats include IGES and DFX.

For more information, contact Micro Engineering Solutions, 32969 Hamilton Court, Farmington Hills, MI 48018; (313) 553-8470.

Reader Service Number 52

Autodesk updates AutoCAD, adds AutoShade

Autodesk has announced version 2.6 of AutoCAD, for delivery in March of 1987. According to the company, the new version extends AutoCAD's three-dimensional capabilities, offers associative dimensioning, and extends AutoLisp to allow access to the symbol tables. The update will retail for \$2850.

Available in the first quarter is AutoShade, a full-color shaded rendering package that allows users to make AutoCAD drawings into solid images showing color, perspective, and surface shapes and features. The program re-

quires AutoCAD. It will retail for \$500.

Autodesk has also announced AutoCAD AEC Mechanical, which customizes AutoCAD for mechanical engineers designing building systems. The program features a graphic interface to duct analysis software, plus a library of industry-standard symbols. AutoCAD AEC Mechanical will be available in the first quarter for \$500.

For more information, contact Autodesk, 2320 Marinship Way, Sausalito, CA 94965; (415) 331-0356.

Reader Service Number 53

FREE!

EXCLUSIVE TIME- AND MONEY-SAVING FEATURES:

- A complete, 96-page catalog for more than 2000 products—many available only from Inmac.
- Easy-to-use: clear, comprehensive descriptions; four-color photos of products in use; invaluable helpful hints.
- Convenient comparative shopping guides—by price, feature and quantity.
- Special sections on new products, CAD/CAM printer and plotter supplies, and the industry's most complete cable guide.
- Same-day shipping, 45-day trial, minimum 1-year guarantee, 10-year history of satisfied customers!

**CALL TOLL-FREE
1-800-547-5444
TODAY!**

Or complete and return the coupon below.

Simply mail your completed coupon to:

inmac™

Catalog Department
2465 Augustine Drive
Santa Clara, CA 95054
Or call

1-800-547-5444

Name _____

Company _____

Address _____

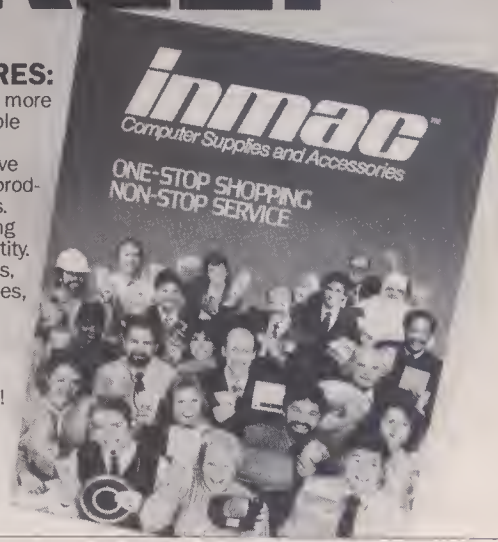
City _____

State _____

Zip _____

Phone () _____

Reader Service Number 8



Borland's Eureka solves complex problems

Borland International's Scientific and Engineering Software Division has announced Eureka: The Solver, a second-generation software productivity tool for IBM PCs. The product is designed to solve complex problems, generate tables and plots, and perform what-if analysis.

The user writes an equation, sets options, and instructs the software to solve the problem. The program can also evaluate and display the accuracy of the solution. Equations, solutions, and evaluations are accessible through separate windows on the screen. According to the company, Eureka supports the 8087 math coprocessor.

Eureka solves equations directly or through the iterative method of successive

approximation. Problems can be solved for simple or complex variables on both sides of the equation, with up to 20 equations solved at a time. The program also solves minimization and maximization problems with or without constraints.

Eureka: The Solver requires an IBM or compatible PC with PC-DOS (MS-DOS) 2.0 or later. It is compatible with CGA, and Hercules and compatible graphics boards. The program is not copy protected. It costs \$99.95, with delivery expected during the first quarter.

For more information, contact Borland International, 4585 Scotts Valley Dr., CA, 95066; (408) 438-8400.

Reader Service Number 54

Adra System's 3000 joins CADD family

Adra Systems has announced the Adra 3000 CADD System. According to the company, the system is a multifunctional, expandable, industry-standard CADD system for mechanical and architectural design and drafting applications. The color system costs \$18,750, while the monochrome system costs \$14,750.

An optional 20M-byte hard disk costs \$1000, while a 100M-byte hard disk with internal 45M-byte cartridge tape drive costs

\$5000. The system comes with 4M bytes of memory that can be upgraded in 4M-byte increments costing \$3250 each up to 12M bytes.

At the same time, Adra announced the first-quarter availability of Unix System V (\$1250), MS-DOS, Ethernet, Tektronix, and an IBM PC-AT Coprocessor (\$2000).

Contact Adra Systems, 59 Technology Dr., Lowell, MA 01851; (617) 937-3700.

Reader Service Number 55

Pyramid Connection connects PCs

Pyramid Technology has announced the Pyramid Connection, a group of products that integrates IBM PCs and compatibles, Apple Macintoshes, and Sun Microsystems workstations into a network for 30 or more machines.

The products include: an Appletalk/Ethernet gateway, Kinetics' Fastpath; Dualport OSx, which implements Appletalk protocols within the OSx kernel; an Appletalk multiwindow terminal, Kinetics' K-term; Centram's Tops for Unix, which provides file access to Macintosh and IBM PC-compatible computers from the Unix host; Centram's Tops for the Macintosh, a distributed file sys-

tem for the Macintosh; Centram's Tops for the IBM PC; Pyramid's Network File System; and PC-NFS, a distributed file system for IBM PCs and compatibles.

Two basic systems, the 98xe and 98x, are available. Prices start at more than \$161,000 for a 98xe configured for PC or Macintosh networking, and around \$245,000 for the 98x configured for PC or Macintosh networking. For more information, contact Pyramid Technology, 1295 Charleston Rd., PO Box 7295, Mountain View, CA 94039-7295; (415) 965-7200.

Reader Service Number 56

Compute server, workstations from Harris

Harris Computer Systems Division has announced a compute server and two engineering workstations.

The Multiple Application Compute Server (MACS) connects with other computer systems through an Ethernet network for processing of engineering data. It supports the Network File System.

The MACS reputedly operates at 7.5 megawhstones, has 16 32-bit registers, cycles in 100 ns, offers 4 gigabytes of virtual memory and 17 gigabytes of disk space, and transfers data at a rate of 80M bytes per second.

A standard MACS configuration, priced at \$285,200, includes 8M bytes of memory, a 330M-byte disk, Ethernet, and NFS. A configuration with 16M bytes of memory costs \$309,200.

The Harris Station 240 and 260 engineering workstations can be connected to the MACS or used as multiuser systems for applications requiring high-resolution graphics. They are 32-bit systems based on Unix and the Motorola MC68020 microprocessor with a built-in 68881 coprocessor. They reputedly operate at up to 3.6 MIPS per processor.

Both workstations include a hi-res, 19-inch, color graphics subsystem capable of displaying 1024 colors from a palette of over 16 million. The HS-260 provides up to 28 bit planes of graphics memory.

The HS-240 costs \$49,900 and the HS-260, \$84,900. For more information, contact Harris, Computer Systems Div., 2101 W. Cypress Creek Rd., Ft. Lauderdale, FL 33309-1892.

Reader Service Number 57

NEW LITERATURE

Fifth Generation Computers. Richard Miller's book analyzes the objectives, research activities, and potential impact of the eight leading fifth generation projects. The Fairmont Press, Inc., 700 Indian Trail, Lilburn, GA 30247; (404) 925-9388; \$45.

1987 Sourcebook. Annual publication of the North American Telecommunications Association, listing voice/data telecom products, vendors of telecommunications systems, and network-building firms. Also contains articles concerning telecom management, new tax policies, and application-feasible technologies. NATA, 2000 M St., N.W., Ste. 550, Washington, DC 20036; (202) 296-9800; \$45.

EDUCOM guidebook. This association of educational institutions interested in information technology has announced publication of its first annual *Guide to Corporate Associates*. The guide includes profiles of 59 EDUCOM-affiliated companies, describing their products, programs, and services. EDUCOM, PO Box 364, Carter and Rosedale Roads, Princeton, NJ 08540; (609) 734-1915; \$14.95.

Second review of PHIGS standard. The Accredited Standards Committee on Information Processing is again releasing for review and comment American National Standard X3.144-198x, which specifies a set of functions for computer graphics programming, the Programmer's Hierarchical Interactive Graphics System—a graphics system for applications producing computer-generated pictures on line or raster graphics output devices. The review period ends January 27. Global Engineering Documents; (800) 854-7179; \$75.

Metabyte's new Fall 1986 handbook. Describes company's line of plug-in boards for data acquisition, industrial control interfaces for PCs, and communications interfaces for IBMs and compatibles. Metabyte Corp., 440 Myles Standish Blvd., Tauton, MA 02780; free.

Engineering Software Report. ESR features surveys and articles on such topics as simulation, project management, forecasting and scheduling, and database engineering. The Center for Technology Management, PO Box 470, Clarkston, GA 30021; (404) 296-2640; \$87.

Interface databook. National Semiconductor interface product line databook describes drivers and receivers, bus transceivers, display controllers, and memory and microprocessor support. C.M.C. Publications, 575 Sinclair Frontage Rd., Milpitas, CA 95035-5470; (408) 945-1557; \$9.

TUTORIAL WEEK ORLANDO '87

HYATT ORLANDO
KISSIMMEE, FLORIDA
MARCH 16-20, 1987

Fifteen Tutorials in 3 Tracks Plus a Workshop

Attendance limited to 100 per tutorial and 24 for the workshop. **SO REGISTER NOW.** To receive the advance registration discount, registration must be postmarked no later than February 20, 1987. Requests for refunds must be received in writing by February 20, 1987. **Registration includes Tutorial Notes and Luncheon.**

TUTORIALS/WORKSHOP

	Track 1	Track 2	Track 3	Workshop
Monday March 16 9 a.m.-5 p.m.	1. Software Configuration Management I Stan Siegel E. Below	2. Knowledge-Based Systems Stephen F. Smith	3. Local Area Networks Ira Cotton	An intensive 5-day workshop on UNIX* and C John Carson Alan Feuer *UNIX is a trademark of Bell Laboratories Inc.
Tuesday March 17 9 a.m.-5 p.m.	4. Software Configuration Management II Stan Siegel E. Below	5. Computers for AI Applications Benjamin W. Wah	6. Parallel Algorithms Narsingh Deo	
Wednesday March 18 9 a.m.-5 p.m.	7. Knowledge-Based Systems for Software Development & Maintenance Allen Goldberg	8. Tools for Building Expert Systems Nancy Martin	9. Concurrent and Distributed Programming Arthur Bernstein	
Thursday March 19 9 a.m.-5 p.m.	10. Automated Tools for Software Engineering Edward Miller	11. The Management of Expert System Development Nancy Martin	12. Performance and Reliability Evaluation of Computer Systems Kishor Trivedi	
Friday March 20 9 a.m.-5 p.m.	13. Software Testing Edward Miller	14. Computer Aided Engineering Workstations J. Hilbert G. Thomas Mitchell	15. Systems Performance Gains with Parallel Processing Techniques Mat Myszewski John Perry	

COMMITTEE

General Chair:

Ratan Guha, Univ. of Central Florida

Program Chair:

Murali Varanasi, Univ. of South Florida

Finance Chair:

Russell Theisen, Martin Marietta Corp.

Treasurer:

Bill Habingreither, Computer Society of the IEEE

Local Arrangements:

Brian Petrasko, Univ. of Central Florida

Publicity & Promotion:

Ed Gallizzi, Eckard College
Frank Fluett, Gould, V.S.O.
Ed Soistman, Martin Marietta Corp.

HOUSING FORM

Tutorial Week Orlando '87

March 16-20, 1987

Hyatt Orlando, Kissimmee, FL

Complete and mail this form to:

Hyatt Orlando, 6375, Kissimmee, FL

Phone 305-396-1234

Reservations must be received by the Hyatt Orlando prior to February 15, 1987. Rooms will be held until 6:00 p.m. on the day of arrival unless accompanied by deposit to cover first night's lodging. A block of rooms has been reserved in the name of Tutorial Week Orlando. Please be sure to mention that you are attending Tutorial Week Orlando. Rooms are available for check-in at 4:00 p.m. on the day of arrival. Check out time is 12:00 a.m.

Name _____

Affiliation _____

Address _____

City/State/Zip _____

Phone No. _____

Please reserve _____ room(s)

☐ \$85 Single ☐ \$95 Double

Credit card type: AX DC CB
VISA MC

Exp. Date _____

Credit Card no. _____

Signature _____

REGISTRATION FEES & FORM

	Early Fees (postmarked by 2-20-87)		Late Fees (postmarked after 2-20-87)	
	Member	Non-Member	Member	Non-Member
All Week Tutorials	\$1000	\$1250	\$1100	\$1350
Per Tutorial	\$210	\$265	\$235	\$295
Workshop	\$1200	\$1500	\$1300	\$1600

I wish to register for the above tutorials/workshop (please circle tutorial/workshop desired).

Total amount due: \$ _____

☐ Check enclosed ☐ VISA/BankAmericard ☐ MasterCard ☐ American Express

Make checks payable in U.S. Dollars to: **Tutorial Week Orlando '87.**

Card No. _____ Expiration Date _____

Signature _____

Name _____ IEEE or IEEECS Membership No. _____

Affiliation _____ No. _____

Address _____

City/State/Zip _____

MAIL TO: **Tutorial Week Orlando '87**, Computer Society of the IEEE, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903, Phone No. (202) 371-1012, TWX: 7108250437 IEEE COMPSO.



THE COMPUTER SOCIETY
OF THE IEEE



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

Recent IC Announcements

MODEL AND FUNCTION	COMPANY	COMMENTS	R.S. NO.
Bt401, Bt402, Bt403, Bt404 Pipelined SRAMs	Brooktree Corp. 9950 Barnes Canyon Rd. San Diego, CA 92121 (619) 452-7580	Designed for use as color palette RAMs in high-performance graphics systems, these four 256 × 8-bit devices feature 4-ns cycle times. The Bt401 (100's price: \$39) and Bt402 (100's price: \$39) offer 3 × 8-bit Sidecar RAMs; they are directly compatible with 10K ECL and 100K ECL, respectively. The Bt403 (100's price: \$36) and Bt404 (100's price: \$36) are functionally equivalent to the Bt401 and Bt402, respectively, but do not feature Sidecar RAM.	70
82A436; 82C435 Enhanced graphics controller; bus interface	Chips and Technologies, Inc. 521 Cottonwood Dr. Milpitas, CA 95035 (408) 434-0600	Fully compatible with the IBM Enhanced Graphics Adapter, the 82C435 (100's price, \$50) offers full backward compatibility to CGA, Hercules, and IBM Monochrome modes. It features 640 × 480 resolution and supports eight color planes. The 82A436 (100's price, \$15) is implemented in a 68-pin PLCC package integrating bus interface logic, including buffers and drivers, memory and I/O select logic.	71
Application-specific IC (ASIC) video controllers	Genoa Systems Corp. 73 E. Trimble Rd. San Jose, CA 95131 (408) 432-9090	Intended for OEMs. Two proprietary VLSI CMOS families are offered. The first (\$30 each in OEM volumes) has the gate array chip designed into Genoa's Spectrum video board products. Features include 132-column and 25/44 line text mode capability. The second (\$50 each in OEM volumes) has the two-chip set designed into Genoa's Super EGA products. A 6845 CRTC is incorporated in the set.	72
HADC77100A, HADC77100B A/D converters	Honeywell, Inc. Signal Processing Technologies 1150 E. Cheyenne Mountain Blvd. Colorado Springs, CO 80906 (303) 577-3511	These are 8-bit, monolithic, flash A/D converters. Both versions digitize a 2-volt analog input signal, with full Nyquist frequency components, at a nominal rate of 150 megasamples per second. The HADC77100A (100's price, \$295) features 70 MHz full-scale bandwidth and ± ½ least-significant-bit linearity; the HADC77100B (100's price, \$200) is available in ± ¼ LSB linearity.	73
HS 9372K, HS 9372J, HS 9372TB, HS 9372SB DACs	Hybrid Systems Corp. 22 Linnell Circle/Suburban Industrial Park Billerica, MA 01821 (617) 667-8700	These are serial input, 16-bit, current output multiplying DACs. The four models, with the monotonicity (over temperature) and temperature range of each, are the HS 9372K, 16 bits min., 0°C to +70°C; the HS 9372J, 15 bits min., 0°C to +70°C; the HS 9372TB, 16 bits min., -55°C to +125°C; and the HS 9372SB, 15 bits min., -55°C to +125°C. They are priced at \$65, \$48, \$150, and \$125, respectively.	74
HY51C256L, HY51256 DRAM	Hyundai Electronics America 4401 Great America Pkwy. 3rd Floor Santa Clara, CA 94043	Fabricated in HYCMOS technology, these 256K × 1 dynamic RAMs feature ripplemode operation. Standard access times available are 100, 120, and 150 ns. The HY51C256L (1000's price, \$5) also features low standby current and a 32-ms extended refresh cycle. The HY51256 costs \$3.50 (1000's).	75
PLHS501 Random Logic Unit Programmable logic device	Signetics Corp. PO Box 3409 Sunnyvale, CA 94088-3409 (408) 991-2000	Based on a single NAND array with internal foldback paths and supported by Signetics' PC-based Amaze design automation software, this PLD implements logic functions utilizing DeMorgan's Theorem. Available in a 52-pin PLCC, the PLHS501 costs \$26.99 (100's).	76
TC511000P-85 DRAM	Toshiba America, Inc. 2692 Dow Ave. Tustin, CA 92680 (714) 832-6300	This 1M-bit × 1 dynamic RAM has an access time of 85 ns. Multiplexed address inputs permit packaging in a standard 18-pin plastic DIP. Features include read-modify-write, CAS before RAS refresh, RAS-only refresh, hidden refresh, and fast page mode capability. Contact a local sales office for 1987 prices.	77
XC 2018 Logic Cell Array Programmable gate array	Xilinx, Inc. 2069 Hamilton Ave. San Jose, CA 95125 (408) 559-7778	The XC 2018 LCA contains the equivalent of 1800 conventional gates on one user-programmable chip. It operates at toggle rates up to 70 MHz and comes in an 84-pin package, or a 68-pin package with a pin-compatible upgrade for users of the XC 2064. The XC 2018 costs from \$15 to \$18 in production volumes.	78

Recent Microsystem Announcements

MODEL AND FUNCTION	COMPANY	COMMENTS	R.S. NO.
Smartport Communications board	Arnet Corporation 476 Woodcrest Ave. Nashville, TN 37210 (615) 254-0646	Board features four or eight RS-232 serial ports, 64K bytes of dual-ported RAM, and an Intel 80186 processor. Compatible with the IBM PC/AT, Compaq 386, and Intel 80386 processor computers, and works with Unix, Xenix, and other multiuser operating systems. Cost: \$895 for Smart-4; \$1295, Smart-8.	80
QPC-5101 Motherboard/graphics adapter/disk controller	Qualogy 2241 Lundy Avenue San Jose, CA 95131 (408) 434-5200	Combination IBM PC/XT motherboard, color graphics adapter, and floppy/Winchester controller. Has 640K bytes of RAM and of EPROM, on-board BIOS, a Harris *CP80C88 CMOS micro, IBM CGA- and monochrome-compatible graphics controller, 3.5- and 5.25-inch floppy controller, and SCSI Winchester controller. Cost: \$1295.	81
PME 68-21, -22, and -23 68020 VME processor boards	Plessey Microsystems One Blue Hill Plaza Pearl River, NY 10965 (914) 735-4661	The 32-bit 68-21 features no-wait state, pre-fetch cache, and VSB interface; the 16-MHz 68-22 offers 68851 paged MMU, SCSI interface, and 2M-byte dual-port RAM; and the 20-MHz, 32-bit real-time 68-23 has multiprocessor capability. Cost: PME 68-21 (with accelerator and computer cards) \$54.65; 68-22, \$44.95; and 68-23, \$16.41.	82
ARCNET-PC310 LAN controller	Standard Microsystems Corp. 35 Marcus Blvd. Hauppauge, NY 11788 (516) 273-3100	IBM PC and compatible board features COM9026 LAN controller and COM9032 ARCNET LAN transceiver with a transmitting medium of 200-μ fiber-optic cable. Has a 2K-byte packet buffer. Cost: \$1295.	83
All Aboard 286 SMT board	IDEAssociates, Inc. 29 Dunham Rd., Billerica, MA 01821 (617) 663-6878	Surface mount technology board for IBM PC AT and XT 286 features 128K, 4M, or 16M bytes of memory (addressable by 80286 machines), serial and parallel ports, and IBM enhanced graphics adapter functions. Cost: 128K-byte memory, \$995; 4M-byte memory, \$2595; and 16M-byte memory, \$12,995.	84
Prose 4000 Voice output board	Speech Plus, Inc. PO Box 7461 Mountain View, CA 94039 (415) 964-7023	Text-to-speech voice module with PCB for IBM PC/XT/AT and compatibles and diskette with I/O drivers for MS DOS. RS-232 port and connector, on-board amplifier, volume control, and audio jack included. Drivers callable from Lattice C and Microsoft C, Basic, and assembler. Cost: \$1750.	85
Evercom-24 Modem card	Everex Systems Inc., 48431 Milmont Dr. Fremont, CA 94538 (415) 498-1111	2400-baud half-card-size modem compatible with Hayes, Bell 212A/103, CCITT V.22, and V.22 bis systems. Designed with adaptive equalization and auto dial, answer, redial, data transfer rate adjustment, and data-to-voice switching. Cost: \$289.	86
GESMPU-20 Processor board	GESPAC 50A West Hoover Ave. Mesa, AZ 85202-5266 (602) 962-5559	Single-height 4- × 6.25-inch Eurocard has 68020 32-bit processor with operating speed of up to 5 MIPS. Equipped with four 32-pin sockets for up to 512K bytes of EPROM and with 256K bytes of CMOS RAM. Cost \$1250.	87
Mighty Meg Memory board	Quadram One Quad Way Norcross, GA 30093 (404) 923-6666	0.5M-byte to 3.5M-byte extended memory board compatible with the Compaq 386 and the IBM AT and 286 XT and compatibles. Features SIMM technology with 256 × 9 DRAM and can be used for RAM disks and protected-mode operating systems. Cost: \$545 for 0.5M bytes; \$1475 for 3.5M bytes.	88
MVME390A Color graphics interface	Motorola PO Box 20912 Phoenix AZ 85036	VMEbus-compatible module has 1024- × 1024-pixel resolution, bit-mapped memory, and pixel addressing. Includes interlaced operation option, TTL-level RGB/monochrome video, and an 384K-byte RAM. VMEbus slave interface has RAM access via 24- or 32-bit address space. Cost: \$2495.	89

CALL FOR PAPERS

Call for papers for Computer

Computer magazine seeks articles that cover the state of the art and important new developments in computer science, technology, and applications. Aimed at a broad audience with diverse interests and experience, *Computer* usually publishes surveys or tutorials that facilitate the transfer of technology from university to industry, from research to applications, and across specialized fields. Submit six copies of the manuscript, including illustrations, references, and authors' biographies, to

- Bruce Shriver
IBM T.J. Watson Research Center
PO Box 704 (H0-B04A)
Yorktown Heights, NY 10598
(914) 789-7626
Compmail + : b.shriver
CSnet: shriver@ibm.com
Vnet: shriver at yktvmh

Computer also seeks articles on topics relevant to upcoming special issues. Submit six copies of the manuscript directly to the guest editor:

- **Systolic Arrays: From Concept to Implementation:** Submit by **February 1, 1987**, to Benjamin Wah, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory, 1101 W. Springfield Ave., Urbana, IL 61801; (217) 333-3516. The issue is scheduled for publication in July 1987.
- **CAD-Based Robot Vision:** Submit by **February 1, 1987**, to Bir Bhanu, MS MN65-2300, Honeywell Systems and Research Center, 3660 Technology Dr., Minneapolis, MN 55418; Arpanet, bhanu@hi-multics; phone, (612) 782-7676. The issue is scheduled for August 1987.
- **Integrated Design and Programming Environments:** A joint issue with *IEEE Software* is in the planning stages. Check next month for details on **March 1, 1987**, submission to the guest editors, Peter Henderson and David Notkin.
- **Standards:** Material is also sought for the "Standards" department. For

details regarding scope and length, contact the department editor: Helen M. Wood, National Bureau of Standards, B154 Technology, Gaithersburg, MD 20899; (301) 975-3240.

Lastly, *Computer* seeks special-issue proposals and articles in the following topic areas:

- optical computing,
- electronic publishing and technology,
- the European computer industry,
- fabrication and packaging technologies,
- noncoded information,
- computer-integrated manufacturing,
- technology transfer process,
- computers in consumer electronics, automobiles, etc.,
- design and implementation of large-scale systems, and
- engineering and scientific workstations.

Prospective guest editors and authors should submit proposals and articles directly to Bruce Shriver.

ACM SIGGraph 87 (IEEE): July 27-31, 1987, Anaheim, California. Submit five copies of the manuscript (20 pages maximum) and a 100- to 300-page abstract by **January 13, 1987**, to Maureen C. Stone, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304.

ASPLOS-II, Second International Conference on Architectural Support for Programming Languages and Operating Systems: October 5-8, 1987, Palo Alto, California. Submit five copies of full-length papers (approximately 5000 words) or short papers (approximately 1000 words) by **February 1, 1987**, to Randy H. Katz, Computer Sciences Division, UC Berkeley, Evans Hall, Berkeley, CA 94720.

☎ **Compsac 87:** October 5-9, 1987, Tokyo. Papers and proposals for panel discussions are sought. Submit five copies of the full-length paper (1000 to 5000 words; include a 150-word abstract). Submit five copies of proposals for panel discussions (include names of proposed panelists and a 150-word

scope statement). Materials in both categories must be received by **February 1, 1987**. Send them to Tosiya L. Kunii, c/o Business Center for Academic Societies Japan, Yamazaki Bldg. 4F, 2-40-14, Hongo, Bunkyo-ku, Tokyo 113, Japan; phone 81 (3) 817-5831 (authors in Asia, Australia, and New Zealand) or Albert K. Hawkes, Sargent & Lundy, Engineering Consultants, 55 E. Monroe, Chicago, IL 60603; (312) 269-3640 (authors in other areas).

☎ **IEEE Transactions on Computers:** Papers are sought for a special issue on supercomputing. Submit six copies of papers by **February 1, 1987**, to H.C. Torng, School of Electrical Engineering, Cornell University, Ithaca, NY 14853-5401; (607) 255-5191. (Guidelines for submitting manuscripts appear on the back cover of every issue of *IEEE Transactions on Computers*.)

VII International Symposium on Protocol Specification, Testing, and Verification (IFIP): May 5-8, 1987, Zurich, Switzerland.

Submit four copies of the complete paper (including abstract) by **February 1, 1987**, to H. Rudin and C.H. West, IBM Zurich Research Lab, 8803 Ruschlikon, Switzerland; phone 41 (1) 72-48-433.

AAAI-87, Sixth National Conference on Artificial Intelligence: July 13-17, 1987, Seattle. Submit six copies of the paper (4000 words maximum) by **February 10, 1987**, to AAAI-87, American Association for Artificial Intelligence, 445 Burgess Dr., Menlo Park, CA 94025-3496.

11th ACM Symposium on Operating Systems Principles: November 9-11, 1987, Austin, Texas. Submit 12 copies of the paper (5000 words maximum) by **February 16, 1987**, to Alfred Spector, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

☎ **ICCD-87, IEEE International Conference on Computer Design: VLSI in Computers and Processors:** October 5-8, 1987, Rye Brook, New York. Submit three copies of a 1000-word summary by **February 16, 1987**, to Prathima Agrawal, AT&T Bell Laboratories, 600 Mountain Rd., Room 3D-480, Murray Hill, NJ 07974; (201) 582-6943.

☎ **1987 Workshop on Visual Languages:** August 19-21, 1987, Linköping, Sweden. Submit three copies of paper (5000

Conferences that the Computer Society participates in or sponsors are indicated by the IEEE Computer Society logo; other conferences of interest to our readers are also included. For inclusion in Call for Papers or Calendar, submit information **six weeks before the month of publication** (e.g., for the **April 1987** issue, send information for receipt by **February 15, 1987**) to **COMPUTER**, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.

CALENDAR

words maximum) by **February 20, 1987**, to Erland Jungert, FFV Elektronik AB, Agatan 29, S-582 22 Linköping, Sweden.

Third Aerospace Computer Security Conference: Applying Technology to Systems (AIAA, American Society for Industrial Security): December 8-11, 1987, Orlando, Florida. Technical papers, ideas for panel discussions, and proposals for tutorials are sought. Unclassified preliminary materials should be sent by **February 28, 1987**. Proposals for papers and panel discussions should be sent to Stephen Walker, Trusted Information Systems, Inc., PO Box 45, Glenwood, MD 21738. Submit proposals for tutorials to J. Chris Perry, Mitre Corp., 1820 Dolley Madison Blvd., McLean, VA 22102. Classified sessions at the secret level are planned. Preliminary unclassified abstracts and proposals for such sessions should be sent to Stephen Walker.

IEEE Workshop on Languages for Automation: August 24-27, 1987, Vienna. Submit papers by **March 1, 1987**, to M. Tauber, Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260.

International Conference on Supercomputing (IFIP, SIAM): June 8-12, 1987, Athens. Submit five copies of the paper by **March 15, 1987**, to Elias Houstis, Dept. of Computer Science, Purdue University, West Lafayette, IN 47907 (authors in the Americas) or T. Papatheodorou, Computer Technology Institute, PO Box 1122, 26110 Patras, Greece (authors in other areas).

Eighth Annual International Conference on Information Systems (ACM, SIM): December 6-9, 1987, Pittsburgh. Submit four copies of the complete paper (25 pages maximum, including illustrations, tables, and references) by **March 30, 1987**, to Charles H. Kriebel, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213.

Performance 87 (ACM, AFCET, IFIP): December 7-9, 1987, Brussels. Submit five copies of a complete paper (20 pages maximum) by **April 13, 1987**, to P.-J. Courtois, Philips Research Laboratory, 2 Ave. E. Van Becelaere, Box 8, B-1170 Brussels, Belgium; phone 32 (2) 673-41-90.

⊕ **ICCAD-87, IEEE International Conference on Computer-Aided Design:** November 9-12, 1987, Santa Clara, California. Submit 12 copies of both a one-paragraph abstract and a more detailed description not to exceed 1500 words or six pages by **May 1, 1987**, to ICCAD-87 Secretary, Mentor Graphics Corp., 1940 Zanker Rd., San Jose, CA 95112; (408) 436-1500.

January 1987

⊕ **IC-87, Symposium on Intelligent Control, January 18-20, Philadelphia.** Contact A. Meystel, Drexel University, Dept. of Electrical and Computer Engineering, College of Engineering, Philadelphia, PA 19104; (215) 895-2220.

⊕ **Syscon, OEM Computer Peripherals Subsystems Conference and Exposition, and Buscon 87 West, Bus/Board Users Show and Conference, January 19-21, Los Angeles.** Contact Edward E. Grazda, 17100 Norwalk Blvd., Suite 116, Cerritos, CA 90701-2750.

⊕ **Fourth International Symposium on Modeling and Simulation Methodology: Intelligent Environments, Goal-Directed Models (SCS), January 19-23, Tucson, Arizona.** Contact Paul A. Baltes, Office of Special Professional Education, University of Arizona, Harville Bldg., Box 9, Tucson, AZ 85721; (602) 621-3054.

⊕ **Annual IEEE Design Automation Workshop, January 21-23, Apache Junction, Arizona.** Contact Jim Armstrong, Electrical Engineering Dept., Virginia Tech, Blacksburg, VA 24061; (703) 961-4723 or (703) 961-7078.

⊕ **IEEE Workshop on Instrumentation for Distributed Computing Systems, January 22-23, Sanibel Island, Florida.** Contact Andre van Tilborg, Computer Science Dept., Carnegie Mellon University, Pittsburgh, PA 15213; (412) 268-3801.

February 1987

⊕ **Third International Conference on Data Engineering, February 2-6, Los Angeles.** Contact Gio Wiederhold, Dept. of Computer Science, Stanford University, Stanford, CA 94305; (415) 497-0685 or Benjamin Wah, Coordinated Science Lab, University of Illinois, Urbana, IL 61801; (217) 333-5216.

CSC-87, 1987 ACM Computer Science Conference, February 17-19, St. Louis. Contact Arlan DeKock, University of Missouri-Rolla, Computer Science Dept., Rolla, MO 65401; (314) 341-4491.

⊕ **18th Annual SIGCSE Technical Symposium (ACM), February 19-20, St. Louis.** Contact Dan C. St. Clair, UMR Graduate Engineering Center, University of Missouri-Rolla, 8001 Natural Bridge Rd., St. Louis, MO 63121-4499; (314) 553-5440.

⊕ **Third IEEE Conference on Artificial Intelligence Applications, February 22-28, Orlando, Florida.** Contact Jan Aikins,

Aion Corp., 101 University Ave., fourth floor, Palo Alto, CA 94301; (415) 328-9595.

Workshop on Distributed Information Systems: Networks and Applications (IEEE), February 23, Washington, DC. Contact IEEE Workshop, 12498 Fan Leaf Court, Fairfax, VA 22033; or D. McDysan, (703) 734-2263; or B. Jabbari, (301) 428-5660.

⊕ **Compcon Spring 87, February 23-26, San Francisco.** Contact Glen G. Langdon, Jr., IBM Dept. K54-802, 650 Harry Rd., San Jose, CA 95120-6099; (408) 927-1818.

Sixth Annual Phoenix Conference on Computers and Communications (IEEE), February 25-27, Scottsdale, Arizona. Contact Forouzan Golshani, Dept. of Computer Science, Arizona State University, Tempe, AZ 85287; (602) 965-2855.

March 1987

⊕ **20th Annual Simulation Symposium (ACM, IMACS, SCS), March 11-13, Tampa, Florida.** Contact James Gantt, 5986 Dana Dr., Norcross, GA 30093; (404) 894-3107.

⊕ **19th Southeastern Symposium on Systems Theory, March 15-17, Clemson, South Carolina.** Contact Ernest Baxa, Jr., Electrical and Computer Engineering Dept., Riggs Hall, Clemson University, Clemson, SC 29634-0915; (803) 656-5900.

⊕ **Tutorial Week Orlando 87, March 16-20, Kissimmee, Florida.** Contact Ratan Kumar Guha, Computer Science Dept., University of Central Florida, Orlando, FL 32816; (305) 275-2341.

⊕ **Sixth Symposium on Reliability in Distributed Software and Database Systems (NASA), March 17-19, Williamsburg, Virginia.** Contact Edwin C. Foudriat, NASA, Langley Research Center, Information Systems Division, MS 469, Hampton, VA 23665; (804) 865-3535.

1987 IEEE VLSI Test Workshop, March 24-25, Atlantic City, New Jersey. Contact Wesley E. Radcliffe, IBM East Fishkill, Dept. 277, Bldg. 321-5E1, Hopewell Junction, NY 12533; (914) 894-4346.

Southcon (IEEE), March 24-26, Atlanta. Contact Electronic Conventions Management, 8110 Airport Blvd., Los Angeles, CA 90045; (213) 772-2965.

⊕ **IEEE Infocom 87, Sixth Annual Joint Conference of the IEEE Computer and Communications Societies: Global Net-**

works—Concept to Realization, March 30-April 2, San Francisco. Contact IEEE Infocom 87, 1730 Massachusetts Ave. NW, Washington, DC 20036-1903; (202) 371-0101; telex 7108250437 IEEECOMPISO.

⊕ **Ninth IEEE International Conference on Software Engineering (ACM), March 30-April 2, Monterey, California.** Contact William Riddle, Software Design and Analysis, 1760 Bear Mountain Dr., Boulder, CO 80303; (303) 499-4782.

⊕ **Fourth International Symposium on Optical and Optoelectronic Applied Science and Engineering (SPIE), March 30-April 3, The Hague, The Netherlands.** Contact Joseph Yaver, SPIE, PO Box 10, Bellingham, WA 98227-0010; (206) 676-3290.

1987 IEEE International Conference on Robotics and Automation, March 30-April 3, Raleigh, North Carolina. Contact Harry Hayman, 738 Whitaker Terrace, Silver Spring, MD 20901; (301) 434-1990.

April 1987

⊕ **Fourth International Workshop on Software Specification and Design (AFCET, Agence de l'Informatique, Alvey Directorate, LCRST-Japan), April 3-4, Monterey, California.** Contact David Marca, Digital Equipment Corp., Intelligent Systems Tech Group, 77 Reed Rd., MS HL02-3/CID; Hudson, MA 01749.

CHI-87, Conference on Human Factors in Computing Systems/GI-87, Graphics Interface (ACM, CIPS), April 5-9, Toronto. Contact William Buxton or Ron Baecker, University of Toronto, Computer Systems Research Institute, Toronto, Ontario M5S 1A4, Canada.

Electro 87 and Mini/Micro Northeast (IEEE), April 7-9, New York. Contact Dale Litherland, Electro and Mini/Micro, 8110 Airport Blvd., Los Angeles, CA 90045-3194; (213) 772-2965.

⊕ **IEEE Computer Society Symposium on Office Automation: Integration, Interconnection, and Use of Personal Computers (NBS), April 27-29, Gaithersburg, Maryland.** Contact Vincent Lum, Dept. of Computer Science, Naval Postgrad School, Monterey, CA; (408) 646-2449.

⊕ **IEEE Symposium on Security and Privacy, April 27-29, Oakland, California.** Contact Virgil Gligor, University of Maryland, Dept. of Electrical Engineering, College Park, MD 20742; (301) 454-8846.

May 1987

⊕ **Eighth IEEE Symposium on Mass Storage Systems, May 11-14, Tucson, Arizona.** Contact Patrick Savage, Shell

Development Co., PO Box 481, Houston, TX 77001; (713) 663-2384.

⊕ **Compeuro 87, International Conference on VLSI and Computers, May 11-15, Hamburg, West Germany.** Contact W.E. Proebster, IBM Germany, Dept. 3280, Sci. Tech., Schoenaichen St. 220, Boebblingen 7030, West Germany.

⊕ **IEEE Workstation Technology and Systems Workshop, May 12-13, Cherry Hill, New Jersey.** Contact Ralph J. Preiss, 12 Colburn Dr., Poughkeepsie, NY 12603.

⊕ **Third International Symposium on VLSI Technology, Systems, and Applications, May 13-15, Taipei, Taiwan.** Contact Ben M. Y. Hsiao, IBM Corp., Dept. D18, Bldg. 707, PO Box 390, Poughkeepsie, NY 12602 or Morris Chang, ITRI, No. 1, Jen Ai Rd., Sec. 2, Taipei, Taiwan.

⊕ **Eighth Symposium on Computer Arithmetic, May 18-21, Como, Italy.** Contact Mary Jane Irwin, Dept. of Computer Science, 333 Whitmore Laboratory, Pennsylvania State University, University Park, PA 16802 or Luigi Dadda, Dept. of Electronics, Piazza Leonardo da Vinci 32, Politecnico di Milano, I-20133 Milan, Italy.

CG International 87, Fifth International Conference on Computer Graphics in Japan (CGS), May 25-28, Karuizawa, Nagano Prefecture, Japan. Contact Toshiyasu L. Kunii, Dept. of Information Science, Faculty of Science, University of Tokyo, Hongo, Tokyo 113, Japan; phone 81 (03) 812-2111, ext. 4116.

⊕ **1987 International Symposium on Multiple-Valued Logic, May 26-28, Boston.** Contact Dan Simovici, Dept. of Mathematics and Computer Science, University of Massachusetts, Boston, MA 02125; (617) 929-7966.

June 1987

⊕ **IFIP Workshop on CAD Engines, June 1-2, Kakaishinlou, Kainkan, Japan.** Contact Tatsuo Ohtuski, Waseda University, Dept. of Elec. and Comm., School of Science and Engineering, 3-4-2 Okubo, Shinjuku, Tokyo 160, Japan; phone 81 (03) 209-3211.

⊕ **Westex-87, Second Western Expert Systems Conference, June 2-4, Anaheim, California.** Contact Westex 87, PO Box 2111, Fullerton, CA 92633-0111.

⊕ **14th International Symposium on Computer Architecture, June 2-5, Pittsburgh.** Contact Zary Segall, Computer Science Dept., Carnegie Mellon University, Pittsburgh, PA 15213; (412) 268-3736.

⊕ **ICCV-87, First International Conference on Computer Vision, June 8-11, London.** Contact Aziel Rosenfeld, University of Maryland, Center for Automation

Research, College Park, MD 20742; (301) 454-4526.

⊕ **IEEE Workshop on Software Technology Transfer, June 11-13, Santa Fe, New Mexico.** Contact Charles Richter, MCC, Software Tech. Prog., 9430 Research Blvd., Austin, TX 78759.

NCC-87, National Computer Conference (ACM, AFIPS, DPMA, SCS), June 15-18, Chicago. Contact AFIPS, 1899 Preston White Dr., Reston, VA 22091; (703) 620-8900.

⊕ **Second Annual Conference on Structure in Complexity Theory, June 16-19, Ithaca, New York.** Contact Alan Selman, College of Computer Science, 161 Cullinane Hall, 360 Huntington Ave., Boston, MA 02215; (617) 437-8688.

NATO Advanced Study Institute on Testing and Diagnosis of VLSI and ULSI, June 22-July 4, Como, Italy. Contact F. Lombardi, University of Colorado at Boulder, Campus Box 425, Boulder, CO 80309; (303) 492-1437 or M. Sami, Dept. di Elettronica, Politecnico di Milano, Piazza Leonardo da Vinci 32, I-20133 Milan, Italy; phone 39 (2) 2399-3516.

NECC-87, Eighth National Educational Computing Conference (ACM, SCS), June 24-26, Philadelphia. Contact Frank L. Friedman, Computer Activities Bldg., Box JA1, Dept. of Computer and Information Sciences, Temple University, Philadelphia, PA 19122; (215) 787-8450.

⊕ **Second International Conference on Computer Applications, June 24-26, Beijing.** Contact Tse-yun Feng, Penn State University, Electrical Engineering East Bldg., University Park, PA 16802 or Oscar N. Garcia, Dept. of Electrical Engineering and Computer Science, Rm. T-637, George Washington University, 801 22nd St. NW, Washington, DC 20052; (202) 676-7175.

⊕ **SIGPlan 87, Symposium on Interpreters and Interpretive Techniques (ACM), June 24-26, St. Paul, Minnesota.** Contact Mark Scott Johnson, Hewlett-Packard Labs, 1501 Page Mill Rd., 3024, Palo Alto, CA 94304-1181.

⊕ **DAC-87, 24th ACM/IEEE Design Automation Conference, June 28-July 1, Miami Beach, Florida.** Contact Design Automation Conference, P.O. Pistilli, MP Associates, 7366 Old Mill Trail, Boulder, CO 80301; (303) 530-4562.

July 1987

⊕ **CAR-87, Second International Conference on Computer-Assisted Radiology (ACM), July 1-4, Berlin.** Contact Michael L. Rhodes, Multi-Planar Diagnostic Imaging, Inc., 2730 Pacific Coast Hwy., Torrance, CA 90505; (213) 539-5944.

CAREER OPPORTUNITIES

RATES: \$12 per line, \$120 minimum charge (up to ten lines). Average six typeset words per line, nine lines per column inch. Add \$10 for box number. Send copy at least six weeks prior to month of publication to: **Carole Porter, Classified Advertising, IEEE COMPUTER Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.**

STATE UNIVERSITY OF NEW YORK AT BUFFALO

The Department of Computer Science is seeking faculty at the level of Assistant Professor to begin Fall, 1987. Applicants must have a Ph.D. in Computer Science, or a related field, and superior research ability.

The University at Buffalo is the largest public university in New York and New England, with approximately 26,000 students enrolled in 93 undergraduate, 101 master's, and 94 doctoral programs, including law, medical, and business schools.

The Department offers BA, BS, MS, and PhD degrees, with controlled undergraduate enrollment. Departmental computing facilities include a VAX 11/780, two VAX-11/750s, six SUNs, six lisp machines and several image processing/graphics systems. Present research areas include artificial intelligence, computer vision, numerical analysis, parallel algorithms, theory of computation, VLSI, etc. The Department is expanding, with additional faculty lines committed annually. Salaries are extremely competitive.

Resumes and names of four references should be directed to: Prof. Sargur N. Srihari, Chairman of Search Committee, Department of Computer Science, 226 Bell Hall, SUNY at Buffalo, Buffalo, NY 14260 (CSnet: srihari@buffalo). SUNY is an affirmative action/equal opportunity employer.

UNIVERSITY OF CALIFORNIA, DAVIS Faculty Positions In Electrical and Computer Engineering/Computer Science

The Department of Electrical and Computer Engineering at UC Davis invites applications for tenure track positions at all ranks in all specialties of electrical engineering, computer engineering, and computer science. Of particular interest are faculty positions in the areas of integrated optics and opto-electronics, computer architectures, artificial intelligence, computer communications, and image processing.

The department, with 40 faculty members and 150 full-time graduate students, is experiencing rapid growth. Our College is the nation's sixteenth largest producer of engineering Ph.D.'s in a University which has the twentieth largest extramural research funding. Salary and benefits are extremely attractive.

Davis is a pleasant, family oriented community, within easy driving distance to Silicon Valley, the Lawrence Livermore National Laboratory, San Francisco, the Pacific Ocean, and the Sierra Nevada Mountains.

We are seeking individuals with strong records of teaching and research with ambitious plans. Senior appointments require outstanding records of achievement; junior appointments

must show evidence of great promise. All faculty are expected to have a strong commitment to teaching at all degree levels, and to demonstrate the ability to attract significant research support. The positions require a Ph.D. degree or equivalent, and are open until filled. Send a resume and the names of at least three references to:

Professor S. Louis Hakimi, Chair
Department of Electrical and
Computer Engineering
University of California
Davis, CA 95616

The University of California, Davis, is an equal opportunity/affirmative action employer.

MEMPHIS STATE UNIVERSITY Department of Mathematical Sciences

The Department of Mathematical Sciences invites applications for tenure track positions at all ranks in all areas of computer sciences for Fall 1987. Preferred research areas include algorithms, parallel and distributed processing, artificial intelligence/cognitive science, and software development. Applicants must have a Ph.D. by September 1, 1987, and a strong potential for excellence in teaching and research.

The Department offers degrees at all levels including the Ph.D., and provides a very favorable research environment in terms of library and computing facilities (including a symbolic Intel Hypercube), teaching load, travel opportunities, etc.

Selection will begin on February 1, 1987. Applications will continue to be accepted until all positions are filled. Applicants should submit a resume and direct three letters of reference to:

Ralph Faudree, Chairman
Department of Mathematical Sciences
Memphis State University
Memphis, TN 38152

An Equal Opportunity/Affirmative Action Employer

GEORGE WASHINGTON UNIVERSITY Tenure-Track Faculty Positions

The Department of Electrical Engineering and Computer Science invites applications for tenure-track faculty positions. We particularly seek persons active in the areas of Computer Science and Communications. Candidates should have an earned doctorate and research experience, with an interest in both teaching and research. Ability to attract research grants and contracts for support of faculty and student assistants is valued. The George Washington University is located in the center of Washington, DC. This metropolitan area sustains the second largest concentration of research and development activity in the United States which creates a continuing demand for rigorously trained engineers as well as broad research opportunities. Our department is the largest department in the School of Engineering and Applied Science with 33 full-time faculty, and a large graduate and undergraduate student body. The department has an annual research budget of close to \$1

million. Send Curriculum Vitae, list of publications and references, to:

Professor Roger H. Lang, Chairman
Department of Electrical Engineering and
Computer Science
School of Engineering & Applied Science
THE GEORGE WASHINGTON UNIVERSITY
Washington, DC 20052

The University is an affirmative action/equal opportunity employer.

HAVERFORD COLLEGE Tenure track position in Computer Science

Haverford College announces a tenure-track opening in Computer Science at the Assistant or untenured Associate Professor level. Candidates should expect to play a leading role in developing an undergraduate program in Computer Science, appropriate to a liberal arts environment. Haverford has close ties to Bryn Mawr and Swarthmore Colleges, and the three colleges plan extensive cooperation and sharing of resources in Computer Science. Haverford expects its faculty to maintain a clear commitment to research as well as teaching at the undergraduate level. Salaries are competitive, and the college is prepared to offer additional support for research equipment and maintenance costs. A Ph.D. degree is expected by September 1987. Curriculum vitae and three letters of recommendation should be sent by February 15, 1987 to Curtis Greene, Chairman, Computer Science Search Committee, Haverford College, Haverford, PA 19041.

Haverford is an Equal Opportunity/Affirmative Action employer.

UNIVERSITY OF ALASKA-FAIRBANKS

The Computer Science faculty invite applications for anticipated tenure-track and visiting faculty positions, to be filled September, 1987, or January, 1988. Rank depends on qualifications. The University of Alaska-Fairbanks is the major research institution in the University of Alaska system and is the site of several world famous research institutes. The Department of Mathematical Sciences offers B.S. and M.S. degrees in Computer Science. The University facilities include a research quality library and a statewide computer network of VAX 8800 and 8600 computers. In addition, the Department has a VAX 11/780, PDP 11/23, several Masscomp color graphics work stations, and smaller computers. Applicants should have a Ph.D. in Computer Science or equivalent. Applicants with research interests in any field will be considered. Some preference will be given to applicants with expertise in operating systems, architecture, and hardware. Salary is competitive and fringe benefits are excellent. Resume and three letters of reference should be directed to Dr. Clifton Lando, Head, Department of Mathematical Sciences, University of Alaska, Fairbanks AK 99775-1110. Inquiries: (907) 474-7117 or FFBML@ALASKA.BITNET. Deadline for applications: March 15, 1987. UAF is an Equal Opportunity/Affirmative Action employer and educational institution.

THE UNIVERSITY OF MICHIGAN
Ann Arbor, Michigan
Faculty Positions Available

in
Electrical Engineering and Computer Science

Applications are solicited for several additional faculty positions at all ranks in various areas of electrical engineering and computer science and engineering. Qualifications include an outstanding academic record, significant involvement in research, a doctorate in electrical or computer engineering or computer science, and a strong commitment to teaching and research. Areas of specialization are not restricted. However, solid-state devices, circuits and microelectronics, optics, communication networks, image processing, VLSI, operating systems, computer architecture, software engineering, artificial intelligence, languages and compilers, as well as robotics and integrated manufacturing, are of particular interest. Please send resumes:

Professor George I. Haddad, Chairman
Department of Electrical Engineering and
Computer Science

The University of Michigan
Ann Arbor, MI 48109-2122

An equal opportunity/affirmative action employer.

UNIVERSITY OF PITTSBURGH
Department of Computer Science
Junior and Senior Faculty Positions

The Department of Computer Science is entering a period of substantial growth. Several tenure track faculty positions are to be filled as soon as possible, before the end of 1987. Others will be allocated subsequently. Responsibilities include research, supervision of graduate student research and graduate and undergraduate teaching. A Ph.D. in computer science or in a closely related field is required. Any area of specialty in computer science will be considered, although preference will be given to programming languages, operating systems, algorithms, and theory of computation. Salary and rank will be commensurate with a candidate's established record of scholarly achievement and/or research potential.

Additional positions, which are joint appointments with other academic units, are available for the following areas: telecommunications, artificial intelligence, and supercomputing.

The Department has twenty full-time faculty members, strong graduate (M.S. and Ph.D.) and undergraduate (B.S.) programs. It has an excellent research library. Departmental computing facilities include a VAX 11/780 (under BSD UNIX), a PDP-11/70 (UNIX), several smaller PDP-11 systems, a variety of microcomputers, several graphics systems, eleven SUN workstations, and a network of Xerox 1100-series (Dandelion) workstations. The research systems are accessible via the Department's Ethernet-compatible LAN. Convenient access is also provided to the extensive general computer facilities of the University as well as to other networks (e.g., CSNET). The Department operates the Research Institute for Parallel and Distributed Intelligence Systems (IPDIS) to provide an environment for innovative research in computer science, and cooperates closely with the Pittsburgh Super-Computing Center, which operates CRAY X-MP/48.

Pitt is an equal opportunity/affirmative action employer.

Please send your resume to: Dr. Shi-Kuo Chang, Chair, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. CSNET Address CHANG@PITT.

CLEMSON UNIVERSITY
Computer Engineering

ELECTRICAL and COMPUTER ENGINEERING at CLEMSON UNIVERSITY is seeking applicants for tenure-track and visiting positions in Computer Engineering program offering BS, MS and PhD degrees. All areas of computer engineering are of interest, with special emphasis on distributed system software and hardware, computer networking and computer architecture. Positions in Electrical Engineering degree program also available, with areas of specialization including microelectronics, reliability, robotics, communications, signal processing, electromagnetics and power.

Applicants should have a Ph.D. and a strong interest in teaching and research. Research Associate positions (BS,MS) are also available. Send resumes to Dr. A. Wayne Bennett, E&CE Department, Clemson University, Clemson, SC 29634-0915. Clemson is an Equal Opportunity/Affirmative Action Employer.

STANFORD UNIVERSITY
Faculty Openings

The Departments of Electrical Engineering and Computer Science have a number of openings for regular and adjunct faculty and research positions. Applications are invited. Applicants should have a demonstrated research ability in one or more areas of computer hardware and software such as VLSI design, computer architecture, operating systems, or programming languages.

All applicants will be considered for positions beginning with the fall quarter 1987.

Stanford University is an affirmative action, Equal Opportunity Employer and welcomes applications from women and minority groups. Please submit, as soon as possible, a detailed resume, including references to Professor John Hennessy, Chairman, Search Committee, Computer Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, California 94305.

DIALOGUE MANAGEMENT ARCHITECT

Engage in fundamental cognitive science and evaluative research in the area of software to support advanced dialogues between humans and machines including the design, implementation, and report of original experiments. Utilize background in psychology, quantitative methods and statistics, artificial intelligence, linguistics (including current research in syntax, semantics, and dialogue analysis), and human comprehension to design and evaluate software that will model the human's comprehension process. Tailor machine queries and responses to determine the user's goals, increase the user's understanding of the machine's capabilities, accommodate or correct user errors, and present results in forms most comprehensible to the user. Demonstrated ability to perform innovative, original research as shown by publications in appropriate fields and successful record as research assistant in a major research university; LISP programming ability, at least equivalent to the construction of one non-trivial system; and ability to evaluate computer systems for their usability and learnability for users in an industrial setting. Applicants must have a Ph.D. in Psychology. Apply at the Texas Employment Commission, TEC Building, Austin, Texas 78778, J.O. #4640628. Ad Paid by an Equal Employment Opportunity Employer.

UNIVERSITY OF MASSACHUSETTS

Faculty positions in Electrical and Computer Engineering. The Department of Electrical and Computer Engineering at the University of Massachusetts at Amherst is one of the largest in New England with 35 faculty and 275 graduate students. The Department invites applications for tenure-track/tenured positions in: All areas of Computer Engineering. Already in place are strong programs in hardware, computer architecture, VLSI, and distributed processing. Computer facilities including CYBER 174 and 175 machines together with a number of networked VAX 11/780, 11/785 and microVAX machines. The department is the recipient of substantial computing equipment for VLSI-related instruction from the Massachusetts Technology Park Corporation (MTPC) and has many ties with computer companies in the Commonwealth providing research, equipment, consulting and teaching support. Candidates for senior positions are expected to have excellent research records and junior candidates must show strong research potential. All candidates must have interests in teaching as well as a strong research orientation. Teaching loads are generally conducive to the establishment or expansion of high-quality research programs. Attractive features of the University of Massachusetts are the quality of life in the town of Amherst and the University's proximity to the commonwealth's high-technology industry. Salaries are competitive and commensurate with experience. Send resume and names of three references to Professor Keith R. Carver, Head, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 by March 15, 1987. The University of Massachusetts is an Affirmative Action/Equal Opportunity Employer.

UNIVERSITY OF SOUTHERN CALIFORNIA

The Department of Electrical Engineering-systems at the University of Southern California announces tenure track positions at all levels in computer engineering. We are especially interested in candidates with background and/or experience in robotics, AI for vision, computer performance and CAD for VLSI. Send a resume and the names of at least three academic references to: Jerry M. Mendel, Chairman, Dept. of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-0781. USC is an equal opportunity/affirmative action employer.

CONSULTING SYSTEMS ENGINEER

Engineer to be principal architect for IMS/LU6.2 interface for large California bank. Will develop specs and design for distributed data processing utilizing IBM's IMS/VS, CICS/VS, TPF2.2 and APPC/PC technologies. Requires BSCS or equivalent and 4 yrs. experience (if no formal degree, 2 yrs. college and 8 yrs. experience); in-depth knowledge of: banking systems requiring 24-hour on-line processing and transaction processing capacities in excess of 50 tps; IMS/VS DB/DC intervals; IMS/VS fastpath internals; IMS/VS MFS and its implementation of DPM; SNA implementation of IMS/VS LU type P, LU type 2.0; broad knowledge of: IMS/VS XRF; CICS; IBM's SNA architecture & LU type 2, LU6.2; SNA implementation of TRFII SLU Type P; CICS/VS; LU type 6.1 and 6.2 and APPC. Must be able to explain highly technical material to less technically adept staffs. 40 hrs./week; \$55K/year. Job and interview site: San Francisco, CA. Send this ad and resume to Job # FC 8892, P.O. Box 9560, Sacramento, CA 95823-0560 not later than 1/30/87.

RUTGERS UNIVERSITY
Computer Engineering

Rutgers University has openings for tenure-track faculty in its newly organized Computer Engineering Program. Applicants with specialization in software engineering, VLSI design and testing, computer graphics, image processing, computer architecture and parallel processing are of particular interest. Excellent computer and research laboratory facilities are available. Teaching loads are modest and reflect the university's objective of building a program with excellence in teaching and research. Applicants are expected to have a strong commitment to both research and teaching. Interested individuals should send a resume and the names of four references to

Professor Herbert Freeman
Computer Engineering Program
Rutgers University
P.O. Box 909
Piscataway, NJ 08854

Rutgers is an equal-opportunity, affirmative-action employer.

UNIVERSITY OF WISCONSIN—MILWAUKEE
Department of Electrical Engineering
and Computer Science

Applications are invited for faculty positions in Computer Science. Candidates should have a Ph.D. in Computer Science or a related discipline. A commitment to research and teaching is expected. Senior candidates should have a strong and established research record. Preferred areas are: programming languages, compilers, theory, systems, and artificial intelligence. Research investigations by our current Computer Science faculty are in the areas of databases, data security, theory of computation, expert systems, computer architecture, parallel computation, and artificial intelligence.

The University of Wisconsin—Milwaukee campus is located near the shores of Lake Michigan and is close to pleasant and beautiful residential areas. Interested persons should send a resume with three references to:

Professor K. Vairavan
Chairperson—Computer Science
Department of Electrical Engineering &
Computer Science
University of Wisconsin—Milwaukee
P.O. Box 784
Milwaukee, WI 53201

UNIVERSITY OF MARYLAND

The Electrical Engineering Department of the University of Maryland College Park has openings for regular faculty positions in computer engineering. Candidates for the rank of Assistant Professor should have a high potential for both teaching and research. Candidates for the rank of Associate Professor and Full Professor should have distinguished records in research and a strong interest in educational programs. The department conducts research and educational programs in all areas of computer engineering. Although applications in all areas of computer engineering will be considered, networking, operating systems, computer security, architectures for parallel computing, and VLSI design are currently the highest priority areas for adding new faculty. Application, including resume, list of publications, list of references, identification of technical area for which the candidate wishes to be considered, should be sent to Dr. William W. Destler, Chairman, Electrical Engineering Department, University of Maryland, College Park, Maryland 20742. The University of Maryland is an equal opportunity, affirmative action employer.

SOUTHERN METHODIST UNIVERSITY
Department of Computer Science and
Engineering

The Department of Computer Science and Engineering invites applications for a tenured senior faculty position and regular tenure-track or visiting faculty positions at all ranks. Applicants for the senior position should have outstanding reputation in academic pursuits and experience in leading group research projects. Applicants for the regular tenure-track positions should have a Ph.D. degree and demonstrated research and teaching experience in computer science, or a related area. Salary will be commensurate with qualification and experience. SMU is a private university with approximately 9000 students. CSE is in the School of Engineering and Applied Science, where a close working relationship exists with the Departments of Electrical Engineering and Operations Research of SEAS, and with Mathematics Department and Business School. The Department has extensive contacts with computer-related and engineering-oriented industrial firms that distinguish Dallas as one of the top centers for high technology, and which has pushed Dallas into major prominence in executive and research lab relocation. CSE presents a balanced program of research and education at all levels and has been offering Ph.D. programs for some twenty years. Our current faculty strength lies in scientific computation and knowledge/data based systems. Preferred specialties for new faculty are computer and system architecture, artificial intelligence, software engineering, and parallel algorithms and systems. Departmental computing facilities include a VAX 11/780, several Sun Microsystems, a PDP 11/70, three LISP machines, numerous personal computers, graphics equipment, and EPROM and IC testing equipment. CSE has recently moved to new expanded facilities, and has been targeted for exceptional growth in the University's new planning document, "The Decade Ahead." Applicants should send a complete resume, including three references to:

D.Y.Y. Yun, Chairman
Department of Computer Science and
Engineering
Southern Methodist University
Dallas, TX 75275

SMU is an equal opportunity/affirmative action employer. Applications from women and minorities are particularly encouraged.

UNIVERSITY OF MIAMI
Electrical and Computer Engineering
Chairperson

The Electrical and Computer Engineering Department of the University of Miami invites applications from qualified candidates in the fields of electrical engineering, computer engineering and computer sciences for the position of Chairperson. Candidates should have an earned doctorate in one of the above fields as well as teaching, industrial or government experience; substantial record of externally funded research; and evidence of leadership ability. The Department, which is the largest in the College of Engineering, has 18 full-time faculty plus 2 open lines, with both undergraduate and graduate (M.S. and Ph.D.) programs. Applications with resumes and names of three references (with addresses) should be sent to Dr. T. Nejat Veziroglu, ECE Chair Search Committee, College of Engineering, P.O. Box 248294, Coral Gables, Florida 33124; phone (305) 284-4666. The application deadline is January 20, 1987. Starting date is no later than August 15, 1987. The University of Miami is an affirmative action/equal opportunity employer.

WEST VIRGINIA UNIVERSITY

ELECTRICAL AND COMPUTER ENGINEERING: The Electrical Engineering Department at West Virginia University anticipates available faculty positions for 1987-88 in the areas of digital and computer engineering systems design, telecommunications, microelectronics, and electromagnetics. (Applicants are requested to identify their field of interest in their cover letter of transmittal of curriculum vitae.) Salary and rank will be commensurate with qualifications. Positions will be tenure track. Applicants must have the Ph.D., potential for high quality teaching and will be expected to initiate research and participate in departmental research programs. Curriculum vitae should be sent to: Dr. Ronald L. Klein, Professor and Chairman, Department of Electrical Engineering, West Virginia University, P.O. Box 6101, Morgantown, WV 26506-6101. Applications should be received by March 31, 1987 although searches for available positions will continue until filled. West Virginia University is an affirmative action/equal opportunity employer m/f.

RENSSELAER POLYTECHNIC INSTITUTE
Computer Engineering Positions Available

Several junior-level, tenure-track faculty positions are available in the computer engineering area, as part of a recently approved plan for growth and expansion in the Department of Electrical, Computer, and Systems Engineering. The Department is the largest academic unit at Rensselaer Polytechnic Institute (RPI) with 45 full-time faculty members and approximately \$4 million in external research. In the 1985-1986 academic year, the Department awarded 303 bachelor degrees, 117 masters degrees, and 18 doctoral degrees. The Department's faculty are engaged in several areas of advanced study and research, including computer engineering, automatic control (including robotics and automation), information and decision systems (including manufacturing systems), communication and signal processing (including telecommunication), electronics and circuits, solid state, and fusion plasmas.

Applicants must have an earned doctorate. Priority is given to applicants who have the following interests and background: computer architecture, computer hardware design, VLSI design, parallel processing, computer performance evaluation, knowledge-based systems and artificial intelligence. Research opportunities exist in association with one of the many Department laboratories or one of the three major research centers (Center for Integrated Electronics, Center for Interactive Computer Graphics, and Center for Manufacturing Productivity and Technology Transfer). Additionally, faculty are encouraged to seek individual research grants and to consult independently. Successful applicants will receive special initial benefits as new faculty members, including summer remuneration, a reduced teaching load, research assistant support, etc.

While this search is primarily for junior faculty, established senior faculty with a strong research and publication record are also encouraged to apply. Opportunities exist for appointment to a Chaired position.

Applicants should send a resume, the names and addresses of four references, and a letter indicating their research and teaching interests to:

Dr. James M. Tien
Acting Department Head
Department of Electrical, Computer, and
Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, New York 12180-3590

RPI is an Affirmative Action/Equal Opportunity Employer

CALIFORNIA STATE UNIVERSITY
San Bernardino

POSITION FOR LECTURER [non-tenured] (\$25,811-\$49,547).

ASST. PROF. (\$31,489-\$37,871).

ASSOC. PROF. (\$36,077-\$43,472).

OR FULL PROF. (\$44,343-\$53,507) of COMPUTER SCIENCE dependent on qualifications and experience.

Duties include teaching, advising, curriculum development, program evaluation and community interaction. Teaching load equated to 12 hrs/wk of lecture and lab. Ph.D. in CSci is preferred, but an MS in CSci with Ph.D. in related field will be considered. Candidates with an MS in CSci working towards a Ph.D. may be considered for a lecturer appointment. Facilities include CDC Cyber 720 and 730-760, Prime 9750, DEC PDP 11 computers, many PCs, microprocessors and terminals.

The University is located in one of the fastest growing service areas in the USA with a current student population over 7500 and more than 300 computer science majors. The area is noted for its warm climate and is within one hour of beaches, mountain resorts and metropolitan Los Angeles without their high living costs. Applicants should submit letter of application, vitae, 3 letters of recommendation and official transcripts. Selection will be completed no sooner than March 15, 1987. Send materials to: Dr. Richard J. Botting, Chair, Department of Computer Science, California State University, 5500 University Parkway, San Bernardino, CA 92407, BITNET:PAAR@CALSTATE.

An Equal Opportunity/Affirmative Action, Section 504, Title IX Employer.

GEORGIA INSTITUTE OF TECHNOLOGY

The School of Information and Computer Science invites applications for faculty positions at all levels. Applicants should have a commitment to teaching and should show a record of outstanding research accomplishments or expect to receive a Ph.D. degree by Fall 1987 and show high potential for research. The School seeks applicants to strengthen its capabilities in all areas of current research activity, especially artificial intelligence, software engineering, distributed computer systems, and also in computer graphics, programming languages, theoretical computer science, human factors, VLSI systems, data communications and computer networks, database systems, and computer architecture. Very competitive salaries are offered.

The School is well supported by the Institute. It has 24 faculty members and is currently experiencing substantial faculty growth. Its educational activities include an undergraduate program accredited by the Computing Sciences Accreditation Board, Inc., a Masters program with 150 students and a Ph.D. program with over 70 students. Well equipped separate laboratories support research and education. High-speed local area networks interconnect all major campus laboratories and provide access to CSNet and other national networks.

Georgia Tech is located in Atlanta, which experiences a mild sunbelt climate. It is the center of commerce in the Southeast, offering a diverse economy and good employment opportunities in all professional areas. Atlanta offers good cultural and recreational opportunities, extremely attractive residential neighborhoods, and affordable housing.

Candidates should send complete resumes and names of at least three references to: Professor Nancy D. Griffith, Chairman, Faculty Search Committee; School of Information and Computer Science; Georgia Institute of Technology; Atlanta, Georgia 30332. Georgia Tech is an equal opportunity employer

COMPUTER PROGRAMMING MANAGER

COMPUTER PROGRAMMING MANAGER to design and customize a computer clinical laboratory software and hardware package which will be used for hospital application. Worker will discuss the requirements necessary with the Chief Technologist and then translate these requirements into a software design on a computer. Worker must have the ability to converse both in the area of chemical and biological assays as well as in the area of computer software languages. In order that the final product can be fully integrated with the existing hospital information systems, worker must have a knowledge of order-entry, accounts receivable, patient accounting, general ledger, materials management, payroll, fixed assets, admission, discharge and transfer subsystems in order that proper interfaces are in the design. Worker will program the software product on the MAI Basic Four computer using knowledge of the hardware in order to optimize the data storage requirements and the execution speed. Upon completion of the programming, worker will oversee the installation, and client training, of the finished product. Bachelor degree with major field of study being in Accounting, Business or Science plus 4 years experience required. \$1,000 per week. Place of employment and interviews: Laguna Hills, CA. Send this ad and your resume to Job #CW9429, P.O. Box 9560, Sacramento, CA 95823-0560 not later than January 31, 1987.

WESTERN WASHINGTON UNIVERSITY

Funding permitting, Computer Science will have two tenure-track openings for the academic year starting Sept., 1987. Ph.D. in Computer Science or closely related field is required. There is need to staff the mainline undergraduate courses and to strengthen the masters' program. For more information, please contact James Johnson, Chairman, Computer Science Dept., Western Washington University, Bellingham, Washington 98225; CSNET address: johnson %www@csnet-relay.arpa WWU is an EO/AA employer.

NEW MEXICO STATE UNIVERSITY
Position Announcement

The Department of Computer Science will have tenure-track position(s) available beginning Fall 1987. Applicants from all areas of computer science are welcome. Qualifications include a Ph.D. in computer science or related discipline and strong potential in both research and teaching.

The department offers B.S., M.S., and Ph.D. programs. Currently there are 12 full-time regular faculty with active research areas including artificial intelligence, computer architecture, programming languages, and operating systems. In Spring 1987, the Department will move into a fine new building which will contain excellent computing facilities connected to a campus-wide network, the Rio Grande Research Corridor network, and CSNet.

The Computing Research Laboratory (CRL) at New Mexico State University is one of five Centers of Technical Excellence established by the state of New Mexico. The CRL provides opportunities for research support and is currently funding computer science faculty and graduate students in the areas of artificial intelligence, networking, cognitive modeling, database systems, and computer vision.

To apply, please send a resume including the names and addresses of three references to Professor Yorick Wilks, Department of Computer Science, Box 3CU, Las Cruces, NM 88003-0099. Position will be open at least until March 15, 1987. AA/EEO Employer

UNIVERSITY OF CALIFORNIA
SANTA BARBARA
Electrical and Computer Engineering

Applications are invited for three tenure track faculty positions in computer engineering. One position is in the area of design automation (CAD tools, techniques, algorithms, or expert systems) to support and/or validate VLSI-based designs. The remaining positions may be filled from any specialty within computer engineering, with preference given to applicants experienced in the fields of parallel computing, very high-speed digital circuit design/analysis, or fault-tolerant computing. The positions start at the rank of Assistant Professor, although higher level appointments are possible for individuals with outstanding records. Normally, completion of a doctorate is required at the time of the appointment. Candidates should have an outstanding research potential or a distinguished research reputation, the ability to attract external research funding, and a strong commitment to teaching at the undergraduate and graduate levels. Applicants should send their resume, copies of recent publications, and the names of at least four professional references to: Chairman, Computer Engineering Faculty Search Committee, Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, (805) 961-3821. Applications will be received until the positions are filled. An Equal Opportunity/Affirmative Action employer.

NAVAL POSTGRADUATE SCHOOL
Monterey, CA
Position Announcement in Computer Science

The Department of Computer Science has immediate openings for faculty positions at all levels in the areas of Artificial Intelligence and Computer Architecture. An applicant should have a PhD in Computer Science or a related field and have a strong interest in both graduate teaching and research. Senior applicants must have distinguished research records. Appointments can begin at any time during the year.

The Department offers MS and PhD degrees in Computer Science supported by well-equipped instructional/research facilities and a full-time technical staff. The faculty normally teach for two quarters and conduct full-time research supported by major research programs during the other two quarters.

Please send a detailed resume and three letters of reference to: Professor Vincent Lum, Chair, Computer Science Department (Code 52), Naval Postgraduate School, Monterey, CA 93943, telephone (408) 646-2449. NPS IS AN EQUAL OPPORTUNITY/AFFIRMATIVE ACTION EMPLOYER.

WASHINGTON UNIVERSITY IN ST. LOUIS

The Department of Electrical Engineering at Washington University in St. Louis is seeking a new Chairman to guide an expansion of about one-third from its present eighteen person level; additional positions are anticipated from retirement over the next decade. A major gift to underwrite the opening phases of this development is already in hand. Persons desiring additional information should write or telephone Professor Fred J. Rosenbaum, Department of Electrical Engineering, Washington University, St. Louis, Missouri 63130 [(314)889-6154]. Washington University is always interested in superior applicants regardless of race, creed, color, gender or age.

CALIFORNIA STATE UNIVERSITY, HAYWARD
Department of Mathematics
and Computer Science

The department is now seeking applicants for tenure track Assistant Professor positions in Computer Science beginning Fall, 1987. Particularly well qualified, experienced individuals with the capability to exercise leadership in the program may apply for appointment at the rank of Associate Professor.

Applicants should have the Ph.D. in Computer Science, or in a related field with computer science experience. They should have a commitment to excellence in teaching and a willingness and an ability to participate in curriculum development. Applicants should also exhibit the competence and potential to engage in significant professional activities, including research and publication. All areas of specialization will be considered. The interests of the present faculty include a wide range of theory, software design, and hardware design topics.

CSU, Hayward is located in the hills above, and overlooking, the eastern shore of the San Francisco Bay. More than 12,000 students attend this university, which has outstanding programs in arts, letters, science, and business. The Department of Mathematics and Computer Science enrolls nearly 1300 majors in its three degree programs: B.S. in Computer Science, B.S. in Mathematics, and M.S. in Mathematics. An M.S. in Computer Science is being planned.

The department has a Pyramid 90x minicomputer and four Sun3-160 color workstations connected on an Ethernet, a graphics lab, an interactive classroom, a digital and a microprocessor lab, and numerous IBM PC's for faculty use. In addition, the campus has a Cyber 170, a PDP 11/44, a PRIME 9720, and numerous PC's for student use.

Interested applicants should send a resume and the names of three references to

William R. Nico, Chair

Department of Mathematics and Computer Science

California State University, Hayward
Hayward, CA 94542

UUCP: . . . ucbvax!lll-crg!csuh!nico

Applications received by February 20, 1987, will be assured full consideration. Applications will be accepted as long as positions remain available.

California State University, Hayward, is an Equal Opportunity/Affirmative Action employer and encourages applications from women and men of all ethnic backgrounds and physical abilities.

UNIVERSITY OF ALBERTA
Department of Computing Science

The Department of Computing Science is undergoing an extensive expansion in research initiatives. Applications are invited for four tenure-track positions at the Assistant/Associate Professor level. Responsibilities include research as well as teaching at the graduate and undergraduate levels. Candidates from all areas will be considered. Current hardware support includes an Amdahl 5870, a network of VAX 11/780's, and well equipped microcomputer and workstation laboratories for graphics, VLSI, and AI research. Access to a Cyber 205 is available. Salary range is \$31,612 to \$50,630 and is commensurate with qualifications and experience. Send curriculum vitae, names of three references, and up to three reprints or papers. New Ph.D.'s should also include a copy of their transcript. Apply to:

Dr. Lee White, Chairman
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1

Applications will be accepted until April 30, 1987. The University of Alberta is an equal opportunity employer.

QUEEN'S UNIVERSITY AT KINGSTON
Department of Electrical Engineering

Applications are invited for a new tenure-track position in the Computer Engineering program of the Department. Applicants should have expertise in one or more of the following areas: microprocessors; computer architecture and systems; computer communications and networks; VLSI and digital systems design. Basic duties will include undergraduate and graduate teaching, research compatible with the departmental program and participation with other staff in the academic and administrative affairs of the department. Rank and salary will be commensurate with qualifications and experience. A Ph.D. degree is required. Candidates of both sexes are equally encouraged to apply. This tenure-track position is provided by the Queen's University National Scholars Program which has been established to promote the highest standard of excellence. In accordance with Canadian Immigration requirements priority will be given to Canadian citizens and permanent residents. Send a detailed career summary with names and addresses of three referees to Dr. G. J. M. Altken, Head, Department of Electrical Engineering, Queen's University, Kingston, Ontario, K7L 3N6. Please quote reference number: Q-16-86.

UNIVERSITY OF CALIFORNIA
SANTA BARBARA

The UNIVERSITY OF CALIFORNIA AT SANTA BARBARA invites applications for four tenure track positions at senior professorial levels in the Department of Computer Science, and for three additional positions in closely related areas, including computer engineering. The Department of Computer Science is in a rapidly expanding College of Engineering that has achieved national prominence in many areas of research. UCSB and the College of Engineering are highly committed to augmenting its Computer Science Department into a department that has both excellence and national visibility. The senior positions in Computer Science currently available are in three major areas of research in which the Department of Computer Science intends to achieve its greatest strength. These areas are: machine vision and motion planning; parallel computing; and software engineering. These positions will provide opportunities for successful candidates to guide the growth and development of the Department in these areas. Strong material support will be given to such positions. "State of the Art" laboratories are currently being planned for instructional purposes in these areas. Research facilities in these areas will be made available and tailored to the needs of successful applicants. Graduate student support will be given high priority.

Interactions with various research groups on campus (Center for Robotic Systems, Cognitive Science Program, Center for Scientific Computing, Institute for Theoretical Physics, etc.) will be strongly encouraged and supported.

Applicants must possess a doctoral degree and an extremely strong record of research accomplishments. Teaching experience is desirable. Outstanding younger candidates are also encouraged to apply. Deadline for receipt of applications is February 15, 1987. Send resume and names of at least eight references to:

Chairman
Computer Science Faculty Search
Committee
Department of Computer Science
University of California
Santa Barbara, CA 93106

The University of California is an Equal Opportunity/Affirmative Action Employer.

WORCESTER POLYTECHNIC INSTITUTE

The Computer Science Department invites applications for tenure track faculty positions at all levels from candidates in all areas of specialization. Candidates should have a Ph.D. in Computer Science and a strong interest in both research and teaching.

Worcester Polytechnic Institute emphasizes quality in the undergraduate learning experience and is committed to an innovative project-oriented teaching environment. The quality of the undergraduate computer science degree has been recognized by the recent granting of accreditation by the Computing Sciences Accreditation Board.

The current goal of the Institute is to enhance our graduate program and improve research activities. The department seeks qualified candidates who will help us achieve these objectives. WPI is located close to the center of Massachusetts' minicomputer industry, and excellent opportunities exist for cooperative research and consulting.

The Department has 12 full-time faculty with 200 undergraduates and 35 graduate students in our M.S. and Ph.D. programs. Department equipment includes three VAX 750's, an MV8000, twenty PDP-11's, and 40 PC's. Much of this equipment is networked via two ethernet cables and is also connected to other campus facilities which include a DEC 2060 and numerous VAX's. The Institute is committed to a new Information Science building and full campus networking facilities in the near future.

Located only 45 miles from Boston, Worcester is a small city of 180,000 which has recently undergone a renaissance. It has eight colleges and universities and a rich variety of cultural activities.

Please send a resume to Prof. R.E. Kinicki, Head, Department of Computer Science, WPI, Worcester, MA 01609.

WPI is an Equal Opportunity/Affirmative Action Employer.

THE UNIVERSITY OF WESTERN ONTARIO
Department of Computer Science

Applications are invited for two tenure track positions at the level of Assistant or Associate Professor in the Department of Computer Science. The Department offers BSc (accredited by the Canadian Information Processing Society), MSc and PhD degrees in Computer Science. Research areas include: artificial intelligence, computer graphics, computer security, design and analysis of algorithms, formal languages and automata, image processing, medical imaging, numerical analysis and software engineering. The Department has in-house facilities (three VAXes), a 24-Sun 3/50 network (with file servers) dedicated to faculty and graduate students, a variety of micros, graphics terminals, two ORCATECH 3050, and a COSMOS imaging machine. The Department can also access on-campus facilities such as a DEC-1091, a Cyber 825, a VAX 8600 and an IBM 4341. Preference will be given to candidates with a PhD (or equivalent research experience) in computer networks and architecture, software engineering, graphics interfaces, expert systems and medical imaging. The appointments are subject to budget approval. Applications and curriculum vitae, including names of three references and up to three papers, should be sent, no later than March 31, 1987, to: Professor I. Gargantini, Chairman, Department of Computer Science, The University of Western Ontario, Middlesex College, London, Ontario, Canada. N6A 5B7. In accordance with Canadian Immigration requirements, priority will be given to Canadian Citizens and Permanent residents of Canada. "An Equal Opportunity Employer."

OHIO STATE UNIVERSITY
Department of Computer and
Information Science

Applications are invited for faculty positions at all levels. A Ph.D. in computer science or a closely related field is required. Of special interest are candidates in the areas of artificial intelligence, computer graphics, data bases, parallel and distributed computing, software engineering, and VLSI design. Special research support packages are negotiable for highly qualified candidates.

Research computing facilities within the Department include DEC 20/60, VAX 11/780, IBM 4341, AT&T 3B2, Xerox Lisp and XDE machines, Sun and HP workstations, and several experimental multi-computers (Intel Hypercube, BBN Butterfly, etc.). In addition to the University's computing facilities (IBM 3081-D, etc.) the department also has access to national networks including ARPANET and CSNET.

To apply, please send application and resume, including a statement of research and teaching interests and the names and address of at least three references, to Prof. Ming T. (Mike) Liu, chairman of Faculty Search Committee, Department of Computer and Information Science, The Ohio State University, 2036 Neil Avenue, Columbus, Ohio 43210-1277 (CSNET/ARPANET:LIU @ Ohio-State.) The Ohio State University is an equal opportunity/affirmative action employer.

THE PENNSYLVANIA STATE UNIVERSITY
Computer Engineering

Applications are invited for tenure-track and visiting faculty positions at all levels. Candidates from all areas of computer engineering (hardware and software) will be considered. The computer engineering program at the Pennsylvania State University is within the Department of Electrical Engineering which has over 50 faculty members, and approximately 1500 undergraduate majors, 170 graduate students. Candidates should have a PhD in Electrical/Computer Engineering or related areas. There are 15 faculty members within the computer engineering program. Excellent instruction and research computing facilities are available within the department, college and at the university computing center.

Please send your letter of application, resume, or inquiries, together with three references to: T. Feng, Computer Engineering Program, Department of Electrical Engineering, 129 Electrical Engineering East, Box C, The Pennsylvania State University, University Park, PA 16802. Deadline for applications is March 31, 1987 or until suitable qualified candidates are selected. An Equal Opportunity/Affirmative Action Employer.

STANFORD UNIVERSITY
Computer Science Department

The department has several openings for tenure track positions in theoretical computer science. The term theoretical computer science is to be broadly construed to include work of a theoretical nature in areas such as programming languages, robotics, graphics and computer-aided design, artificial intelligence, etc., in addition to the more classical areas of algorithms and complexity. Applicants must have a record of strong research and an interest and competence in teaching.

Candidates should send curriculum vitae with suggested references to Prof. Leonidas J. Guibas, c/o Phyllis Winkler, Computer Science Department, Stanford University, Stanford, CA 94305. A deadline of January 31, 1987 is suggested for fullest consideration. Stanford is an Equal Opportunity/Affirmative Action employer and welcomes applications from women and minorities.

UNIVERSITY OF MARYLAND
Information Systems

The Information Systems department has a tenure-track position available at the Associate or Assistant Professor level. Applicants with an interest in both research and teaching are being sought. Attractive salaries are offered.

Information Systems is an active, growing department at the University of Maryland. The graduate program attracts quality students for M.S. and Ph.D. degrees in Information Systems. Current research in our Database Systems Research Center includes topics in database systems optimization, database design, office information systems, knowledge based systems, and systems analysis and design.

For advantageous consideration, applications should be received by February 1, 1987. Candidates should send complete resumes and names of three references to: Prof. Alan R. Hevner, Search Committee Chairperson, Information Systems, College of Business and Management, University of Maryland, College Park, MD 20742, (301) 454-3260. The University of Maryland is an Equal Opportunity, Affirmative Action Employer. Women and minorities are encouraged to apply.

BUCKNELL UNIVERSITY

Bucknell University invites applications for tenure track faculty positions at the assistant or associate professor level. A Ph.D. in Computer Science or related discipline is normally required. Applicants should demonstrate both an interest in and a promise of developing excellence in teaching and research. All research areas considered. First year research and summer support funds available.

The computing environment includes VAX 11/780 VMS, VAX 11/750 UNIX, Sun3 and Apollo workstations, DG Eclipse network lab, HP Instrumentation lab, VAX Station II, and a Honeywell Level 66 DPS mainframe.

Please send a curriculum vitae and the names of three references to Gary Haggard, Dept. of Comp. Sci., Bucknell Univ., Lewisburg, PA 17837 or HAGGARD@BUCKNELL.BITNET. Applications will be considered beginning November 15. Appointments for January 1987 are possible. Applications from women and members of minority groups are encouraged.

UNIVERSITY OF CALIFORNIA, SANTA CRUZ
Computer Engineering
Computer and Information Sciences

The boards of Studies in Computer Engineering and Computer and Information Sciences at the University of California, Santa Cruz, invite applications for positions as Assistant, Associate or Full Professor, as appropriate, effective July 1, 1987. The Division of Natural Sciences at UCSC is expanding its program in Computer and Information Sciences and its new program in Computer Engineering. UCSC currently offers the B.A., M.S., and Ph.D. in Computer and Information Sciences, the B.A. in Computer Engineering, and is developing a graduate program in Computer Engineering.

UCSC is the University of California campus nearest to "Silicon Valley" and is developing close ties with local industry in the computer field. Faculty salaries are competitive, and opportunities for consulting are extensive. UCSC has several outstanding science departments, which have significant interests in scientific computing, including physics and astrophysics, data acquisition and reduction in astronomy (with the Lick Observatory and the Keck telescope) and seismic data processing with the UCSC Richter Seismology Laboratory. AREAS OF INTEREST: Outstanding candidates

from any specialty within computer engineering or computer science are encouraged to apply for these positions. Of particular interest are candidates in computer science with strengths in (one or more of) the following areas: artificial intelligence; database and expert systems; theory and analysis of algorithms (especially algorithms for language processing, scheduling, graphics and image processing, and parallel computation); and in programming languages and compiler construction. In computer engineering, candidates in areas such as software engineering; computer and system architecture; workstations, servers and networks; special purpose architectures (as in signal and image processing); computers as components in larger systems; and in computer-aided design; circuit and system modeling, testing, and performance evaluation, are of particular interest.

RANK: Associate or Full Professor (Please refer to #63-867 in your reply.)

MINIMUM QUALIFICATIONS: Ph.D. in Electrical Engineering, Computer Engineering, Computer Science, or equivalent, and a solid research record as evidenced by publications in technical journals. Applicants will be evaluated on their research record, teaching, professional activities and demonstrated leadership in their field. Industrial experience will be favorably considered.

SALARY: Commensurate with qualifications and experience.

EFFECTIVE: July 1, 1987

TO APPLY: Send applications, including curriculum vitae with cover letter, teaching evaluations, copies of any recent publications, and at least five letters of recommendation evaluating your scholarly contributions, teaching, and other professional accomplishments to the address below.

CLOSING DATE: January 15, 1987.

RANK: Assistant Professor (Please refer to #64-867 in your reply.)

MINIMUM QUALIFICATIONS: Ph.D. in Computer Engineering, Computer Science, or equivalent and demonstrated potential for research and teaching.

SALARY: Commensurate with qualifications and experience.

EFFECTIVE: July 1, 1987

TO APPLY: Send applications, including curriculum vitae with cover letter, and at least three letters of recommendation evaluating your scholarly contributions, teaching, and other professional accomplishments to the address below.

CLOSING DATE: February 27, 1987

SEND APPLICATIONS TO:

Chair, Computer Faculty Search Committee
Applied Sciences Building
University of California
Santa Cruz, California; 95064
UCSC IS AN EEO/AA EMPLOYER

FACULTY FOR EUROPE AND ASIA

Planning a sabbatical or leave of absence? The University of Maryland University College seeks excellent lecturers for undergraduate computer science, computer applications, and information systems management courses on U.S. military bases in Europe and in Asia and the Pacific. Renewable annual appointments begin August 1987. Minimum requirements include a master's degree in computer science or a related field, recent college teaching experience, and U.S. citizenship. Benefits include transportation and military base privileges. Frequent travel and the cost of schooling make these positions difficult for those with children. Send resume to Dr. Lois A. Mohr, Overseas Programs, The University of Maryland University College, College Park, MD 20742-1642. AA/EEO.

THE UNIVERSITY OF MISSOURI-ROLLA
Chairperson
Department of Computer Science

The University of Missouri-Rolla invites nominations and applications for Chairperson of the Department of Computer Science.

The University and the State of Missouri are committed to excellence in Computer Science education and research at the University of Missouri-Rolla. This has been demonstrated by an excellent undergraduate program which is accredited by the CSAB, by the only Ph.D. program offered by public institutions in Missouri, and by the establishment of an Intelligent Industrial Systems Program and an Artificial Intelligence Institute. The department has extensive computing facilities for research in artificial intelligence, real time computing, networking and interactive graphics.

A successful candidate will have an established research record, skills and experience in obtaining external funding from governmental and industrial sources, administrative skills, and the experience to set goals and important directions for the department. A Ph.D. in computer science or a closely related field is required. Salary is very competitive.

The University is situated on Interstate 44 within 2 hours driving time from St. Louis, Springfield, Jefferson City and Columbia in a pleasant non-urban environment. The area is rich in outdoor recreational opportunities.

Nominations and applications should be sent to: Dr. Billy E. Gillett, Chairman of Search Committee, Department of Computer Science, University of Missouri-Rolla, Rolla, Missouri 65401. Applications and nominations will be considered beginning December 1, 1986 and as received until the position is filled. Equal Opportunity/Affirmative Action Employer.

DUKE UNIVERSITY
Department of Computer Science

The Duke University Department of Computer Science, a 1983 recipient of an NSF CER Grant, has faculty positions available at all ranks. Applications are solicited from all areas of computer science. Applicants for senior positions must demonstrate excellence in research, while applicants for junior positions must exhibit the promise of excellence.

The Department currently has seventeen tenure track faculty, approximately 300 undergraduate majors and 50 graduate students pursuing master's and/or doctoral degrees.

The Department has major research efforts in scientific computing with emphasis on numerical linear algebra, the solution of PDEs, and VLSI simulation: computer systems with emphasis on computer architectures, modeling of fault-tolerant systems, systems performance, and communications; artificial intelligence, particularly in the areas of natural language interface, search methodologies, and expert systems; and theory and algorithms with emphasis on combinatorial and graph-theoretic studies. Special motivation for the research efforts comes from the areas of medical applications (in collaboration with the Duke Medical Center), and VLSI (in collaboration with the Microelectronics Center of North Carolina, of which Duke is a Participating Institution).

Interested applicants should send copies of their resumes and other supporting material to

Professor Donald J. Rose
Department of Computer Science
Duke University
Durham, NC 27706

Duke University is an affirmative action, equal opportunity employer.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY
Chairman, Computer Science

Applications are invited for the position of Department Chairman of Computer Science. Applicants should have a doctorate in Computer Science or a related discipline (Computer Science preferred), a record of successful teaching computer science, strong leadership and communication skills, and a commitment to excellence in undergraduate teaching.

Rose-Hulman Institute of Technology, located on a 300 acre campus near Terre Haute, Indiana, is a privately endowed school of engineering and science with 1300 carefully selected students including 100 national merit scholars and 30 graduate students. The department, currently with five faculty members plus shared services from the electrical engineering, physics and mathematics departments, offers a strong computer science program with considerable electrical engineering and mathematics content to approximately 100 C.S. majors and some 25-30 EE/CS and MA/CS double majors. A graduate program at the masters level is planned for the near future as well as application for a Computer Science Accreditation visitation. Computer facilities available to the department include an Institute VAX780 with 40 terminals, numerous PCs and a planned computer science local network of workstations/MicroVAX-II. About half of the student body own PCs.

Initial screening of candidates will begin February 15, 1987 and continue until the position is filled with appointment to be effective September 1, 1987. Rank and salary will be commensurate with qualifications and experience. Send application and resume with three references to Dean James Eifert, Rose-Hulman Institute of Technology, 5500 Wabash Avenue, Terre Haute, Indiana 47803. Rose-Hulman is an equal opportunity employer.

PORTLAND STATE UNIVERSITY
Head of Computer Science Department

Portland State University seeks an outstanding individual with a good understanding of the broad spectrum of computer science, who can promote quality teaching, active research and interaction with industry. An earned Ph.D. in computer science or a related field is required. Candidates should have the ability to work effectively with faculty, students, and administration. Salary and rank are negotiable and commensurate with experience and qualifications.

Portland State University, one of the three major universities in the Oregon State System of Higher Education, is located in Portland, Oregon, a beautiful city at the confluence of the Columbia and Willamette rivers, offering within an hour or two drive a diversity of recreation—skiing and mountain hiking—the unspoiled Oregon coast—unmatched State campgrounds.

At Portland State University, approximately 15,000 students are enrolled in a number of comprehensive academic programs at both the undergraduate and graduate levels, including several doctoral programs. The Department of Computer Science has nine full time equivalent faculty and over 400 full- and part-time majors. The University places special emphasis on the expansion of the computer science program, including continuing evolution of the undergraduate curriculum and development at the graduate level. The department head is expected to lead this undertaking.

The University is entering a new era of dynamic growth and development. During the past three years, the computer and electronics industry, the State of Oregon, and the City of Portland have launched major initiatives to expand computer science and electrical engineering education and research. Portland has a rapidly growing computer and electronics industry in the "sili-

con forest," including Tektronix (world headquarters), Intel, Floating Point Systems (headquarters), Hewlett-Packard, Oregon Software, Sequent Computer Systems, Metheus, Mentor Graphics, etc., which permits close industry-university interaction. Of particular interest, researchers at the nearby Oregon Graduate Center and the Applied Research Laboratory at Tektronix are accessible for interaction and collaboration.

Computing equipment available in the Computer Science Department includes a VAX 11/780 running Unix, several micros and workstations in addition to an IBM 4381 and a Gould 9000 administered by the Campus Computing Center.

The position is available effective July 1, 1987. Forward nominations and applications, including resume and the names of three references, by February 28, 1987 to Professor M.E. Balogh, Chairman, Computer Science Department Head Screening Committee, c/o Dean's Office (EAS), School of Engineering and Applied Science, Portland State University, Portland, OR 97207.

An Equal Opportunity/Affirmative Action Employer.

WORCESTER POLYTECHNIC INSTITUTE
Head of the Computer Science Department

Worcester Polytechnic Institute invites applications and nominations for the position of Head of the Computer Science Department. The Institute seeks an individual who can provide innovative and dynamic leadership of its educational and research programs. This outstanding scientist and educator should have an earned doctorate and a distinguished record of research and scholarly achievement, as well as demonstrated leadership in one of the component specializations of Computer Science. In addition to administrative responsibilities, the department head will be charged with strengthening overall program quality and fostering scholarly excellence, in cooperation with other departments, government and industry.

There are 15 full time equivalent CS faculty, 260 undergraduates, and 50 graduate students. Degrees are offered at the B.S., M.S., and Ph.D. levels. The WPI plan, a project oriented curriculum, forms the basis of a unique and successful undergraduate program that has been accredited by the Computer Science Accreditation Commission of the Computing Sciences Accreditation Board. Department laboratories contain three VAX 750's, twenty PDP 11's, a DG MV/8000, and an IBM Series/1 with 35 PC's, connected by Ethernets. The college is committed to the construction of a new Information Sciences building. Located near Boston, in the center of the computer industry, Worcester has eight universities and colleges, and offers a rich variety of professional and cultural activities.

Nominations and applications, including curriculum vitae and the names of three references, will be accepted until February 15, 1987 or until a suitable candidate is found.

Please respond to:

Professor Robert A. Peura, Chairman, Computer Science Search Committee, Worcester Polytechnic Institute, Worcester, MA 01609. An Equal Opportunity/Affirmative Action Employer.

AURORA UNIVERSITY

Applications are invited for a faculty position in computer science with responsibilities to manage and teach within a new master degree program. Candidates should have a Ph.D. in computer science. AU is located within the high tech corridor west of Chicago. Current hardware support includes a VAX 11/780. Send resume, transcripts and three letters of reference to Homer E. Easley, Dean, School of Information Science, Aurora University, Aurora, IL 60506. AA/EOE

UNIVERSITY OF VICTORIA
Department of Computer Science

Applications are invited for tenure-track positions at all levels to commence July 1, 1987. Applicants should have a Ph.D. in Computer Science or equivalent research achievement. The department has 14 full-time faculty, with several additional positions planned for the near future. It offers graduate and undergraduate degrees in computer science and maintains an active research program. The main research areas are programming languages, compilers, software engineering, distributed computing, fault detection, theory of computation, artificial intelligence, and numerical methods. The computing facilities available for instructional and research support currently include a VAX 11/750, a VAX 11/780, a Pyramid 90X, 10 Sun workstations, microcomputer laboratories with a variety of small systems, as well as the University IBM 3083 system. Also associated with the department is the software engineering laboratory established by a cooperative agreement between IBM Canada and the University of Victoria to advance the state of the art in software engineering.

The city of Victoria, situated on the southern tip of Vancouver Island, has a population of over 200,000. It enjoys one of the most delightful environments in North America. The climate is temperate and all kinds of outdoor activities, from marine to mountain, are popular and can be pursued year round.

Applicants should send a curriculum vitae and the names of three referees to:

Dr. Jon C. Muzio, Chairman,
Department of Computer Science
University of Victoria
P.O. Box 1700
Victoria, B.C., Canada
V8W2Y2

Applications will be accepted until February 15, 1987. Further information is available through electronic mail (JMUZIO@UVUNIX.UVIC.CDN@UBC.CSNET or JMUZIO@UVUNIX.BITNET) or from the department, (604) 721-7220. Canadian Immigration regulations now require the university to assess applications from Canadian citizens and permanent residents of Canada before assessing applications from other persons. The University of Victoria offers equal employment opportunities to qualified male and female applicants.

COLORADO SCHOOL OF MINES

Faculty positions open for computer scientists in an applied program focused on software development, software systems, artificial intelligence, languages and database management. Applicants should hold a Ph.D. and display a strong record or potential for both teaching excellence and research. Rank and competitive salary commensurate with qualifications.

Computer Science shares a department with classical applied mathematics, applied statistics, and operations research. Some of the departmental faculty are world renowned and the institution holds an excellent reputation for education and research in engineering and applied science, especially in relation to minerals and energy.

Applicants should send their resume and reference letters to Ardel J. Boes, Department of Mathematics, Colorado School of Mines, Golden, CO 80401.

Toll free numbers: outside Colorado (1-800-446-9488, x 3882); inside (1-800-245-1060, x 3882). CSM is an Affirmative Action/Equal Opportunity Employer.

TEXAS A&M UNIVERSITY
Department of Computer Science

Applications are invited for faculty positions at the Assistant, Associate or Full Professor level. Candidates from all areas of computer science will be considered. Of particular interest are areas central to the design of fifth generation computer systems. In addition to regular faculty positions, an endowed chair in computer architecture is to be filled.

The Department of Computer Science has 20 full-time equivalent senior faculty members and additional teaching staff. There is a tradition of quality instruction at the B.S., M.S. and Ph.D. levels. The university has initiated a major commitment to develop excellence in Intelligent Systems, Artificial Intelligence, Software Technology, Simulation and other areas of Computer Science. This commitment is in support of the rapid build up of microelectronics and computer technology in Texas, and in particular the decision of the Microelectronics and Computer Technology Corporation (MCC), the industry-wide consortium of computer manufacturers interested in fifth generation computer systems, to locate nearby.

Computer Science at Texas A&M University is located in the College of Engineering. With nearly 10,000 students, the College of Engineering is the nation's largest. Computer Science is expanding to become one of the largest and more comprehensive departments in the country.

We seek excellence in research. Texas A&M will provide superior instructional and research facilities for its Computer Science faculty. The University has the internal resources to achieve this goal. Applicants at the assistant professor level should show substantial promise for research and teaching. Applicants at the higher level should show a strong record of research achievement. Ability in teaching graduates and undergraduates is also essential. All appointments are made on a 12 month basis and academic salaries are highly competitive.

Applicants should submit a resume and three references to: Glen N. Williams, Head, Computer Science Department, Texas A&M University, College Station, TX 77843.

Texas A&M University is an equal opportunity/affirmative action employer.

WAYNE STATE UNIVERSITY
Department of Electrical
and Computer Engineering
Chairperson

The Department of Electrical and Computer Engineering at Wayne State University is seeking applicants with a strong leadership, teaching, and research background for the position of Departmental Chairperson. The Department and College, which are housed in a new building, are committed to a strong program that involves the addition of a substantial number of faculty and new facilities. Significant salary and program development funds are available. Wayne State University is a large urban university located in metropolitan Detroit. It is one of the major research universities in the state of Michigan and its faculty have access to ongoing research and development programs at research centers of the major automobile companies and other industrial organizations in the area. The Department has nearly 1000 students, over 200 of which are in MS and PhD programs. The position will be filled on or before July 1, 1987. Applications, including the names of three references, and nominations should be sent to Dr. Pieter K. Rol, Chairman of the ECE Chair Search Committee, College of Engineering, Wayne State University, Detroit, MI 48202, no later than February 28, 1987. Wayne State University is an equal opportunity/affirmative action employer.

THE UNIVERSITY OF TEXAS AT ARLINGTON

The Department of Computer Science Engineering (CSE) at the University of Texas at Arlington (UTA) invites your application for tenure-track or visiting faculty positions in all areas of computer science and computer engineering. Rank is open. An earned doctorate or equivalent and commitment to teaching and scholarly research is required. Openings are expected for June, or September 1987. Our department offers baccalaureate, masters, and doctoral programs. Excellent computing facilities are available including personal computers in each CSE faculty office. UTA is in the Dallas/Fort Worth metropolitan area near a large concentration of high technology industries. Interested persons should send a resume to Professor Bill D. Carroll, Professor and Chairman, Computer Science Engineering Department, P.O. Box 19015, The University of Texas at Arlington, Arlington, TX 76019. Phone (817) 273-3785.

The University of Texas at Arlington is an Equal Opportunity Affirmative Action Employer.

UNIVERSITY OF CALIFORNIA, IRVINE
Department of Electrical Engineering

The Department of Electrical Engineering at the University of California, Irvine, invites outstanding individuals to apply for tenure track positions at all levels in the area of computer engineering. The levels of these appointments may be at Assistant Professor, Associate Professor, or Professor depending on accomplishments and stature in the field. Subareas of special interest include, but are not limited to

1. parallel structured architectures
2. artificial intelligence
3. software engineering
4. computer graphics

Responsibilities include research and teaching at the graduate and undergraduate levels.

The Department has research activities in fault-tolerant computing, design automation, distributed computing, and real-time computing, and in closely related areas including digital signal processing, image processing, pattern recognition, robotics, digital communication, and communication networks. The Department currently has nineteen faculty members, two of which are members of the National Academy of Engineering and seven Fellows of the IEEE.

Send your complete resume with three references to Professor Jack Sklansky, Chairman of the Computer Engineering Faculty Search Committee, Department of Electrical Engineering, University of California, Irvine, California 92717. The University of California is an affirmative action/equal opportunity employer. To ensure consideration, apply before March 15, 1987.

CLARKSON UNIVERSITY

Faculty Position in Computer Science, Clarkson University. The Clarkson University Mathematics and Computer Science Department has a tenure track position available in Computer Science. Rank and Salary are open. Ph.D.'s with an interest in both research and teaching are being sought. The Department has its own minicomputer system, a VAX 11/750. In addition, the university operates an IBM 4341, a VAX 11/780, and a number of Pro 350 personal minicomputers, running UNIX. Presently all incoming freshmen are being provided with their own personal computers, a Zenith Z-200. Curriculum vitae should be sent to Athanassios S. Fokas, Chairman, Department of Mathematics and Computer Science, Clarkson University, Potsdam, NY 13676. An Equal Opportunity/Affirmative Action Employer.

SYRACUSE UNIVERSITY
Dean
School of Computer
and Information Science

Syracuse University invites nominations and applications for the position of Dean of the School of Computer and Information Science. The Dean will lead an expanding faculty, currently 20 in number, active in a broad program of research with major concentrations in computational logic, programming language theory, and coding and information theory. There are also about 30 affiliated faculty in other units of the University. The School has approximately 200 undergraduate, 100 master's-degree, and 40 Ph.D. students on the main campus, and also offers graduate programs at three extension sites.

The School will move to a new 130,000-square-foot building in September, 1988. Other units, including the Center for Advanced Technology in Computer Applications and Software Engineering (the CASE Center) and the twenty-member computer engineering faculty, will also move to the new building. The School is associated with the research programs of the CASE Center and the Northeast Artificial Intelligence Consortium, and is one of the units involved in the founding of a center for the study of parallel architectures. Candidates should have an established reputation in research and some experience in academic administration. They should be committed to the advancement of computer science and to excellence in scholarship and teaching. The position requires the vision to plan and the leadership ability to direct the School's recruitment efforts, expansion into new research areas, and procurement of external funds.

Syracuse University, a private institution, has major programs of research, graduate studies, and undergraduate education. There are 11,800 undergraduate and 4,400 graduate students on the main campus.

Application and Nomination Procedures. Applicants should send resumes, names of references, and selected publications to: Dean Search Committee, School of Computer and Information Science, 313 Link Hall, Syracuse University, Syracuse, New York 13244-1240, by January 19, 1987. Letters of nomination should be sent by the same date to the above address. The appointment is expected to be effective no later than September 1, 1987.

Syracuse University is an equal opportunity, affirmative action employer.

NORTH CAROLINA STATE UNIVERSITY

The Department of Computer Science invites applications for junior level tenure track positions. Applications are welcome from all areas, but are especially encouraged in the areas of operating systems, artificial intelligence, database systems, programming languages, architecture, and parallel processing. Candidates must have a Ph.D. in computer science or a related discipline and have a strong interest in both research and teaching.

Our department has research ties to organizations in the Research Triangle Park including MCNC, the Microelectronics Center of North Carolina. On campus we have research opportunities with several nationally recognized centers.

Applicants should submit a resume including research interests and three references to:

Dr. Edward W. Davis
Recruitment Committee
Department of Computer Science, Box 8206
North Carolina State University
Raleigh, NC 27695-8206

Immigration status of non-US citizens must be stated in the application. NCSU is an equal opportunity/affirmative action employer.

CENTRAL MICHIGAN UNIVERSITY
Chairperson
Computer Science Department

The Department of Computer Science at Central Michigan University is seeking applicants for the position of chairperson. Responsibilities of the chairperson include all aspects of administration of the department, teaching, development and strengthening of the undergraduate and graduate programs, supervision of graduate students, and research development and support. Applicants must have a doctorate (or equivalent degree) in computer science or a related field. Candidates are expected to have demonstrated expertise in teaching, development of research, and administrative skills in a computer science setting. Salary and rank will be commensurate with qualifications.

The department offers bachelor's and master's degrees in computer science. Computing facilities include a Cyber 170-730, a Burroughs XE550 running UNIX, and IBM PC and Macintosh micro labs. In addition, the University has just committed itself to a major upgrade, including a new mainframe and substantial increases in the number of micros. Faculty teaching and research interests include languages, compilers, database management systems, computer graphics, pattern recognition, software engineering, operating systems, artificial intelligence, computer vision, algorithms and complexity, and networks. The department has about 20 FTE positions, with teaching assignments of 15 hours per academic year (ordinarily three courses one semester and two the other). The chairperson has a nine hour per year teaching assignment.

The University is located in a community of about 24,000, 60 miles north of Lansing, at the edge of Michigan's lakes, hills and woodlands. The student enrollment has held steady at about 16,000 for several years. The University offers outstanding faculty benefits, including a 12% fully paid TIAA-CREF contribution.

Qualified applicants should submit a detailed resume and three letters of reference to

Dr. Jerald A. Kabell
Acting Chairman
Department of Computer Science
Central Michigan University
Mt. Pleasant, MI 48859

Applications will be accepted until March 1, 1987 or until the position is filled, whichever is later. Central Michigan University is an Affirmative Action/Equal Opportunity institution. Women and minorities are especially encouraged to apply.

LOUISIANA STATE UNIVERSITY
Faculty Openings

The Department of Electrical and Computer Engineering at LSU invites applications for tenure-track and visiting faculty positions available August 1987 in all areas of computer engineering, including microprocessors, distributed processing systems and special purpose architectures. A PhD or equivalent and potential for excellence in teaching and research are necessary. Rank is open. Salary is competitive and commensurate with qualifications and experience. Release time and resources are provided in order to enhance the development of a quality research program. Opportunities for summer support are available. Send resume, names of three references and a statement of teaching and research interests to: Alan H. Marshak, Chairman, Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803-5901. LSU is an Equal Opportunity Employer.

UNIVERSITY OF ROCHESTER
Faculty Position
Management of Information Systems

Assistant/Associate/Full Professor of Computer and Information Systems. Will engage in graduate teaching, research, dissertation supervision and related duties in the computer and information systems area in the University of Rochester's William E. Simon Graduate School of Business Administration. Strong training in information systems (MIS, Computer Science or related areas) required; complementary training in economics or operations research desirable. Will do advanced teaching in the information systems area and research leading to publications in leading scholarly journals in all areas of information systems management including the Design and Analysis of Systems, Economics of Information Systems Management, Database Management Systems, Artificial Intelligence and Expert Systems, Telecommunications and Distributed Information Systems. Ability to teach MBA-level courses in information systems required. The leadership of the School in the Management of Information Systems has been recently recognized by its winning a \$2-million grant in IBM's national competition. The School provides an excellent research environment and encourages interdisciplinary work. Faculty salaries and research support very competitive. Send vitae and papers to Professor Halm Mendelson, William E. Simon Graduate School of Business Administration, University of Rochester, Rochester, NY 14627. An Equal Opportunity Employer (M/F).

WAYNE STATE UNIVERSITY

The Computer Science Department invites applications for several tenure track or visiting positions at all levels. Applicants should have a Ph.D. in Computer Science or a related discipline and show a strong commitment to research in an area of Computer Science. The salaries will be competitive and commensurate with the qualifications of the candidates.

Wayne State is one of the research universities in the State of Michigan. It is located in the cultural center of Detroit and serves the educational needs of about 30,000 students. The Computer Science department has a full-time faculty of 17, offers B.S., M.S. and Ph.D. degrees, and has a current total of 190 graduate students (35 Ph.D.). The current major areas of research in the department include database management systems, computer vision, distributed and parallel computing, software engineering, artificial intelligence, and molecular computers.

The department research facilities include approximately 20 SUN and micro VAX workstations, VAX 11/780, and COMPTON vision system, all interconnected by Ethernet.

The university computing center operates an AMDAHL 470/V6, an AMDAHL 470/V8 and several microprocessor labs. The university computer systems are accessible via Telenet and Autonet, and are part of the MERIT network. There is a growing cooperation with research centers of the major automobile companies and other industrial organizations in the area.

Send a Curriculum Vitae and the names, addresses, and telephones of at least three referees to Dr. Horst F. Wedde, Search Committee, Computer Science Department, Wayne State University, Detroit, MI 48202; tel. (313) 577-0731, 577-2477.

Wayne State University is an Affirmative Action/Equal Opportunity Employer. We are particularly interested in minority candidates.

BOWDOIN COLLEGE
Chairperson, Computer Science

The Department of Computer Science & Information Studies seeks a person to lead the development of a young discipline at a distinguished undergraduate liberal arts college. To teach two courses per semester. Commitment to excellence in undergraduate teaching and in research expected. Ph.D. in computer science preferred; applicants with Ph.D. in related field and sufficient computer science experience will be considered. Rank and tenure open.

Bowdoin College is a private, coeducational institution of 1350 students with highly selective admissions and a long tradition of excellence in scholarship. It is located in the greater Portland area along the coast of Maine in Brunswick, a community of 18,000.

College computing facilities include many microcomputers, a DEC-1091 system, and a VAX 11/780 running UNIX.

Salary and fringe benefits are competitive.

To apply, send vitae and the names of three references to Dr. Alfred H. Fuchs, Dean of the Faculty, Bowdoin College, Brunswick, Maine 04011. Applications will remain open until position is filled; review of candidates will begin January 30, 1987. Bowdoin College is committed to Equal Opportunity through Affirmative Action.

MACROMOLECULAR WORKBENCH PROJECT

Department of Pharmaceutical Chemistry, School of Pharmacy, University of California, San Francisco, seeks individuals at the Postdoctoral and Assistant Research levels with experience in numeric, symbolic or graphical computing and structures and interactions of biological molecules, energy calculations, and semi-empirical modelling, to join an active, exciting and well-equipped research group (R. Langridge, I.D. Kuntz, P.A. Kollman, F. Cohen, K. Dill, T.E. Ferrin, R. Ababanel) developing and using computational methods to predict protein tertiary structure from amino acid sequence. Send letter of application and resume with names of three references to Professor Robert Langridge, Computer Graphics Laboratory, University of California, San Francisco, CA 94143-0446. UCSF is an Equal Opportunity Affirmative Action Employer. Women and minorities are encouraged to apply.

UNIVERSITY OF HOUSTON

Applications are invited for tenure track faculty positions in the Department of Computer Science starting September 1987. Areas of special interest include but are not limited to artificial intelligence, computer architecture, operating systems, programming languages and software engineering. Rank and salary are open and competitive. The Department is interested in expanding its research program and particularly welcome applicants for senior positions. Applicants should have Ph.D. in Computer Science or a closely related area, and a strong commitment to research and teaching. Candidates for senior positions should also have a distinguished research record. The Department offers Ph.D., M.S., and B.S. in Computer Science. Departmental research facilities include a network of VAX 11/780 and VAX 11/730's, a network of AT&T 3B20 and 3B2's along with access to other computer facilities in the University Computer Center as well as supercomputers via remote access terminals. Send resume and names of professional references to Dr. Willis K. King, Chairman, Department of Computer Science, University of Houston, Houston, Texas 77004. An Equal Opportunity/Affirmative Action Employer.

ALLEGHENY COLLEGE

The Department of Computer Science is seeking qualified candidates for a tenure track position beginning in September 1987. Applicants should have a Ph.D. in computer science or in a related discipline (Ph.D. candidates will be considered) and should have an interest in teaching and research at a liberal arts college. The computer science program is a rigorous one with courses in both the core and applications of computer science. The program has been accredited by the Computer Science Accreditation Board. Rank and salary are competitive and commensurate with background and experience. We are looking for a candidate able to demonstrate flexibility and breadth as well as mathematical sophistication. This is an ideal opportunity for a person interested in teaching in an accredited computer science program and who also wants the environment of a private liberal arts college. Applicants should submit resume, graduate transcript, and three letters of recommendation to Dr. Benjamin D. Haycock, Department of Computer Science, Allegheny College, Meadville, PA 16335. Applications will be considered until the position has been filled or a decision has been made not to fill it. Allegheny College is an equal opportunity employer.

RESEARCH POSITIONS

ISI, a 15 year old autonomous research institute within the University of Southern California, has several openings in its research programs. Programs have recently begun in integrated man-machine interfaces, uniform interfaces to multiple systems, knowledge representation, expert system diagnosis of aircraft engines, and construction of intelligent user support environments. Ph.D. in Computer Science or related field, or equivalent experience is required. Send resume to: Ms. Lisa Moses, USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292. USC/ISI is an Equal Opportunity/Affirmative Action Employer.

**The Department of Computer Science
at
OREGON STATE UNIVERSITY
and
TEKTRONIX, INC.
proudly announce the new
Tektronix Graduate Fellowships
in
Computer Science**

We are pleased to announce that we anticipate that two new graduate fellowships in Computer Science at Oregon State University will be awarded, beginning with the fall quarter of 1987. These fellowships will carry a stipend of \$10,000 per year.

Recipients will be selected on a competitive basis, with undergraduate performance, scores on the Graduate Record Examination, and references as the primary sources of information. We expect that recipients will enroll in the PhD program in Computer Science at OSU and will devote their full time toward the pursuit of that degree.

For further information, and for application materials, please contact:

Walter G. Rudd, Chairman
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331
503-754-3273

Rud@oregon-state

Oregon State University is an Affirmative Action/Equal Opportunities employer and complies with Section 504 of the Rehabilitation Act of 1973.

THE UNIVERSITY OF CONNECTICUT
Computer Science and Engineering

The Department of Computer Science and Engineering at the University of Connecticut is seeking highly motivated and talented faculty members to fill tenure track positions. The Department offers B.S.E., M.S., and Ph.D. degrees in Computer Science within the School of Engineering. Openings exist at the Assistant and Associate Professor levels while applications from exceptional candidates for Full Professorships are welcome. Applications for visiting faculty positions will also be entertained. Women and minorities are particularly encouraged to apply. Preference will be given to researchers with expertise in artificial intelligence and computer architecture or digital systems, although all qualified applicants will be considered. A full range of computing equipment including a CSNET link provides an environment conducive to research.

The University is located in a rural area in Northeast Connecticut within easy driving distance of several major metropolitan areas. Applicants with Ph.D.'s in Computer Science or equivalent areas with an interest in joining an emerging department are invited to submit resumes and names of three references to: Fred Maryanski, Head, Computer Science and Engineering Department, University of Connecticut, 260 Glenbrook Road, Box U-155, Storrs, CT 06268. An equal opportunity employer. (Search #7A165)

ELECTRICAL ENGINEERING

The University of Missouri-Rolla Engineering Center in St. Louis invites applications for a tenure-track position in the area of Electrical Engineering. A doctorate and research interest are required. Desired areas of specialization include Digital System Design and Application, Control Systems, Computer Aided Design, Artificial Intelligence, Communication Systems, and Software Engineering. Responsibilities include a combination of graduate teaching, research and supervision of graduate students. St. Louis is a desirable living area with numerous cultural and entertainment opportunities. Center courses and degree programs are designed for evening participation by professionals employed in the St. Louis high-tech community. Opportunities exist for interaction with companies and agencies in the St. Louis area. Applications including curriculum vitae and names, telephone numbers, and addresses of three references and nominations, should be sent to Dr. Edward C. Bertnolli, Director, UMR Engineering Center, 8001 Natural Bridge Road, St. Louis, MO 63121. Application deadline is February 1, 1987, or until the position is filled. UMR is an affirmative action, equal opportunity institution.

SYSTEMS ENGINEER

Develop, implement and maintain application programs for inventory management systems for fastener distributors/government contractors. Analyze limitations and capabilities of existing systems and propose resolutions for customers' requirements. Perform accuracy analysis and performance evaluations. Plan new installation or modification of development systems. MS in Computer Science/Ind. Engg., familiarity with Material Requirement Planning Systems and one year experience required. Must be able to use TI-990, 1500, Bus-Pro/Zenix C, IBM 36/PC, RM COBOL computers and software. 40 hours per week. Interview and Job site in San Diego. Send this ad with resume to Job #CW 10105, P.O. Box 9560, Sacramento, CA 95823-0560, not later than February 2, 1987.

LOUISIANA STATE UNIVERSITY
Faculty Position in Image Processing

The Department of Electrical and Computer Engineering and the Remote Sensing and Image Processing Laboratory invites applications for an anticipated faculty position. The person would teach computer vision and computer engineering courses in the Department. Research in computer vision would be conducted with the Remote Sensing and Image Processing Laboratory (RSIP). The position is attractive regarding a teaching load balanced to provide the ability to develop an academic program and provide time and resources for developing a high quality research program in computer vision. The RSIP Laboratory has well qualified staff and modern computing and image processing equipment. A number of peripherals including Symbolics, Sun, and International Imaging workstations are available. A comprehensive image processing system has been implemented in Berkeley Unix 4.3. The appointment will be made at a rank appropriate with the qualifications of the applicant. Interested applicants should apply by submitting a resume, statement of interests, and names and addresses of three professional references. Applications that apply by January 31, 1987 will be given preference. Apply to: Professor Charles A. Harlow, Director, RSIP Laboratory, 150 Electrical Engineering Building, Louisiana State University, Baton Rouge, Louisiana 70803-5901. LSU is an Equal Opportunity Employer.

HARVEY MUDD COLLEGE
Engineering and Computer Science

Applications are invited for a tenure track position in the Engineering Department. Appointment at the Assistant Professor level is anticipated. Applicants should have an engineering background with graduate study in computer engineering or electrical engineering and computer science and experience in both hardware and software design and development. Responsibilities will include teaching in a unified engineering curriculum, developing courses and supervising industrially sponsored projects in the Engineering Clinic. Continuing professional growth and development through research or consulting is expected; excellent opportunities exist in the local area. A doctorate in engineering or computer science is required. Industrial experience is desirable. Reply to: Professor James E. Monson, Chairman, Search Committee, Harvey Mudd College, Claremont, CA 91711. Harvey Mudd College is an equal opportunity/affirmative action employer.

UNIVERSITY OF WISCONSIN—MADISON
Faculty Positions

The Department of Electrical and Computer Engineering invites applications for tenure and tenure-track faculty positions. A Ph.D. degree is required, and successful candidates are expected to participate in both teaching and research activities. Applicants in all areas of computer engineering are invited to apply, but the following areas are of special interest: computer architecture, computer networks, large-scale computational methods, real-time computer applications, design automation, VLSI circuit design, and engineering applications of artificial intelligence. Rank and salary will be commensurate with qualifications and experience. Send resume and names of three references to: J. Leon Shohet, Chairman, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, 1415 Johnson Drive, Madison, WI 53706, an equal opportunity/affirmative action employer.

**THE CENTER FOR
ADVANCED COMPUTER STUDIES
RESEARCH FACULTY
GRADUATE FELLOWSHIPS/ASSISTANTSHIPS
in COMPUTER SCIENCE/ENGINEERING
University of Southwestern Louisiana**

The Center for Advanced Computer Studies is a research center with programs leading to MS/PhD degrees in Computer Science and Computer Engineering. External grants/contracts support research in a variety of areas. The Computing Research Laboratory includes a 40-node Sun-3 network, a 6-processor Encore Multimax, 2 VAX 11/780s, logic development systems, laser printers, plotters, and numerous other equipment. Instruction utilizes a 3-processor Pyramid 90X network running UNIX and an IBM 3090 with a vector processor. Another well-equipped facility supports research in CAD/CAM. About 300 students are enrolled in computing graduate programs.

RESEARCH FACULTY: Applications are invited from persons holding PhD degrees with demonstrated research capabilities in Computer Science/Engineering. Persons appointed as Assistant Professors must hold PhDs in Computer Science/Engineering and have research potential. Associate Professors and Professors must hold PhDs and have an established research publication and grant record. The typical teaching load is 2 graduate-level courses per year and a continuing research seminar. Salaries range from about \$50,000 to over \$100,000 per year. Excellent support is provided for travel, equipment, research assistants, and professional activities. Thus, the necessary environment exists to achieve your professional goals. To apply, send a copy of your curriculum vitae and the names and addresses of at least 3 professional references. Applications will be considered until all positions are filled.

PhD FELLOWSHIPS: A number of PhD Fellowships valued at more than \$14,000 per year are available. They provide support for up to 4 years of study towards the PhD in Computer Science or Computer Engineering. Recipients also receive preference for low-cost campus housing. Completed applications must be received by 15 February 1987.

MS/PhD ASSISTANTSHIPS: More than 100 teaching and research assistantships are available to support students pursuing an MS or PhD degree in Computer Science or Computer Engineering. Stipends are valued at from about \$6,000 up to \$12,000 per academic year. Completed applications must be received by 1 March 1987.

APPLICATIONS: Dr. Terry M. Walker, Chairman, Faculty Search Committee, The Center for Advanced Computer Studies, Lafayette, LA 70504-4330.

An Affirmative Action/Equal Opportunity Employer.

**UNIVERSITY OF NORTH CAROLINA
AT WILMINGTON**

Faculty Position in Computer Science, University of North Carolina at Wilmington, Math. Sciences Department, Wilmington, NC 28403-3297. Douglas D. Smith, Chair (919 395-3291) Asst./Assoc. Professor (rank and salary commensurate with qualifications). Begins August, 1987, Ph.D. in computer science or Ph.D. in a related area with M.S. in computer science or equivalent experience. Any specialty considered. Dept. Faculty of 30 offers bachelor's degrees in computer science and in pure and applied math. VAX/VMS and UNIX are in use. Duties include teaching, scholarship and service. Deadline: Feb. 16, but applications will be received until the position is filled. An EE/AA employer.

DEPARTMENT LEADER

The Department of Electrical Engineering at Washington University in St. Louis is seeking a new Chairman to guide an expansion of about one-third from its present eighteen person level; additional positions are anticipated from retirement over the next decade. A major gift to underwrite the opening phases of this development is already in hand. Persons desiring additional information should write or telephone Professor Fred J. Rosenbaum, Department of Electrical Engineering, Washington University, St. Louis, Missouri 63130 [(314)889-6154]. Washington University is always interested in superior applicants regardless of race, creed, color, gender or age.

UNIVERSITY OF ALBERTA
Department of Computing Science

Applications are invited for a Faculty Service Officer, which is a full time academic position, as DIRECTOR OF INSTRUCTIONAL LABORATORIES, in the Department of Computing Science. Responsibilities include the planning and supervision of instructional laboratories and the promotion of innovations in instructional methodology. Candidates should have considerable relevant academic experience: curriculum planning, teaching experience, course development and supervisory experience. An M.Sc is mandatory but a Ph.D. in Computing Science is preferred. Salary will be commensurate with qualifications and experience; currently salary floor is \$39,620. Departmental instructional laboratory facilities currently include four time-sharing terminal labs connected to the University Amdahl 5870 mainframe computer, three Macintosh PC Labs operated stand-alone or connected to the mainframe, a logic lab, and a micro-computer lab. A VAX 11/780 is largely devoted to several undergraduate courses, running under a UNIX operating system. Applications, including a detailed resume and the names of three academic references should be submitted to:

Dr. Lee J. White, Chairman
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1

The closing date for applications is April 30, 1987 or earlier if a qualified candidate is found. The University of Alberta is an equal opportunity employer.

UNIVERSITY OF WASHINGTON

The University of Washington expects openings for junior and/or senior tenure-track faculty appointments in Computer Science starting in the 1987-88 academic year. A moderate teaching load allows time for both quality research and close involvement with students. As a consequence, we expect applicants to have a strong commitment to both research and teaching and an outstanding record of research for their level. A senior level applicant should have a record that would provide significant new research strength for our department. We invite applicants from all areas of Computer Science, quality being our most important selection criterion. We would particularly welcome applicants with research strength in artificial intelligence, programming languages and compilers, computer systems, and databases. In addition to the above tenure-track positions the department may have a number of visiting positions; these would require both teaching and research, and could be for any portion of the 1987-88 academic year. Interested applicants should send a letter of application, a resume, and the names of four references to Paul Young, Chairman, Department of Computer Science FR-35, University of Washington, Seattle, Washington 98195. The University of Washington is an Affirmative Action/Equal Opportunity Employer. The Ph.D. is required for these positions.

THE JOHNS HOPKINS UNIVERSITY
Department of Computer Science

The Johns Hopkins University has formed an independent Department of Computer Science within the G.W.C. Whiting School of Engineering. During the next five years, a faculty of approximately sixteen will be developed to represent the University in the field of computer science and engineering according to Hopkins' traditional standards of research and teaching of the highest rank. Construction of a new building to house the department will begin during the fall of 1986. Additional faculty are sought at all professorial levels with research and teaching interests in artificial intelligence, computer vision, graphics, programming languages, systems (especially parallel and distributed computing and networks), and theory. Senior candidates should have an outstanding record of research achievement; junior candidates should exhibit superior research potential. Although the teaching load is moderate, genuine commitment to teaching excellent graduates and undergraduates is also necessary.

Applicants should send a resume and the names of at least three references to: Professor Gerald M. Masson, Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218 [phone: (301) 338-8775]. The Johns Hopkins University is an equal opportunity, affirmative action employer.

THE UNITED STATES
NAVAL ACADEMY

Applications are invited for tenure-track positions in the Computer Science Department. An earned Ph.D. in Computer Science or related field is required. The applicant must have a strong interest in combining teaching with research in a very high quality undergraduate program. Previous teaching experience is preferred.

red. Applicants should be able to teach upper division mainstream computer science courses. Civilian faculty receive Excepted Civil Service appointment with initial appointment normally for three years. Salary commensurate with overall experience. Excellent opportunities for research in the local area.

The Computer Science Department offers a major in Computer Science leading to a Bachelor of Science degree. The curriculum follows the latest recommendations of the ACM Curriculum Committee.

Interested persons should send their vitae to: Chairperson, Computer Science Department, United States Naval Academy, Annapolis, Maryland 21402 (phone (301) 267-3080).

Deadline for receipt of applications is 1 February 1987; however, applications will be considered until the positions are filled.

The United States Naval Academy is an Affirmative Action Equal Opportunity Employer.

YORK UNIVERSITY
Faculty of Arts

Department of Computer Science. Two tenure track positions at the Assistant, Associate, or Full Professor level. A Ph.D. in Computer Science, Electrical Engineering or the equivalent is required. One position in computer architecture and one in computer network architectures and protocols. All areas of computer architecture will be considered. Areas of interest in network architecture are formal specifications of network protocols, network protocol verification, network and protocol modelling, architectural performance analysis, international standardization of communication architectures, and the software architectures of network implementations. Duties include research and undergraduate teaching. A Master's program is currently under development. The Department's equipment includes a VAX 8600 (VAX/VMS) and a VAX 11/750

(BSD UNIX), and Sun Workstations, as well as microcomputer and graphics laboratories. Faculty and students also have access to IBM 4381 (VM/CMS) facilities. York's Department of Computing Services engages in a variety of research and development projects in circuit and packet switched data communications. The network testing and development equipment, including a CBX II 9000 with ROLMbus 295, will be used to support academic research whenever appropriate. York University is located in Metropolitan Toronto and within easy reach of downtown Toronto which offers excellent cultural, recreational, and entertainment facilities. Send curriculum vitae and names of three references to Prof. E. Arjomandi, Chairman, Department of Computer Science, Faculty of Arts, York University, North York (Toronto), Ontario M3J 1P3. York University is implementing a policy of employment equity. Qualified women and men are invited to apply. In accordance with Canadian immigration requirements, priority will be given to Canadian citizens and permanent residents of Canada.

ITHACA COLLEGE

The Dept. of Math/CS has two tenure eligible positions beginning Fall 1987. A Ph.D. in CS, Math, or a related discipline required. One position is in CS with rank open. Another is at the Assistant Professor level and selected candidate will be expected to teach at least half the teaching load in CS. Strong preference will be given to applicants who can teach a variety of CS courses. Candidates with primary interest in math and interest in teaching some upper level CS courses are also encouraged to apply. Send vitae to Dr. Eric E. Robinson, Chair, Department of Mathematics and Computer Science, Ithaca College, Ithaca, New York 14850. An EO/AA Employer.

Some New Definitions for

HARDWARE DESIGN ENGINEERS

PC's Lim•it•ed (*noun*) - America's most technologically innovative manufacturer of personal computers. Makers of the fastest and most powerful products available. A company that pays top salaries and offers outstanding creative opportunities.

PC's Lim•it•ed Hard•ware De•sign En•gi•neer (*noun*) - a fully competent EE who is familiar with the Intel 8086, 80286, and 80386 along with the IBM PC family and who enjoys working on the leading edge of design innovations.

Aus•tin (*noun*) - a beautiful city in the Texas Hill Country where 53% of the population is between the ages of 18 and 34, 29% of the residents have a college degree, 34% of household incomes exceed \$50,000, and where cultural and recreational opportunities abound.

Send (*verb*) - to convey, as in "Send your resume to Karen Motsinger, Employment Representative, PC's Limited, 1611 Headway Circle, Building 3, Austin, TX 78676."



PC'S LIMITED

An Equal Opportunity Employer M/F

UNIVERSITY OF VICTORIA
Computer Science

Applications are invited for the position of Chairperson. The appointment is to commence July 1, 1987. The ideal applicant would have a strong record of research and teaching in computer science as well as the potential for academic leadership. The Department has 14 full-time faculty, with several additional positions planned for the near future. It offers graduate and undergraduate degrees in Computer Science and maintains an active research program. The main research areas are programming languages, compilers, software engineering, distributed computing, fault detection, theory of computation, and numerical methods. The computing facilities available for instructional and research support currently include a VAX 11/750, a VAX 11/780, a Pyramid 90X, 10 SUN workstations, microcomputer laboratories with a variety of small systems, as well as the University IBM 3083 system. Also associated with the Department is the Software Engineering Laboratory established by a cooperative agreement between IBM Canada and the University of Victoria to advance the state of the art of software engineering.

The City of Victoria, situated on the southern tip of Vancouver island, has a population of over 200,000. It enjoys one of the most delightful environments in North America. The climate is temperate and all kinds of outdoor activities, from marine to mountain, are popular and can be pursued year round.

Applicants should send a curriculum vitae and the names of three referees to:

Dr. A.T. Matheson
Dean, Faculty of Science
University of Victoria
P.O. Box 1700
Victoria B.C. Canada V8W 2Y2

Applications will be accepted until January 31, 1987 or until the position is filled. Further information may be obtained by telephoning the Department at (604) 721-7220, or through electronic mail (e-mail addresses are jmuizio%uvunix.uvic.cdn@ubc.csnnet or jmuizio@uvunix.bitnet). Canadian immigration regulations now require the University to assess applications from Canadian citizens and permanent residents of Canada before assessing applications from other persons. The University of Victoria offers equal employment opportunities to qualified male and female applicants.

WASHINGTON UNIVERSITY, ST. LOUIS
Regular Faculty Positions in
Computer Science

The Department of Computer Science is expanding its research program and invites applications for several regular (tenure-track) faculty positions at the Assistant, Associate and Full Professor levels. Applicants should hold the Ph.D. or D.Sc degree in Computer Science and have a strong commitment to and record of accomplishment in research.

The Department of Computer Science has a well-respected undergraduate program and a small, but growing graduate program. Research activities have flourished in recent years and have three foci: concurrent systems, communications systems, and intelligent computer systems. Support for these research programs is from NSF, NIH, ONR, DMA and a large number of industrial sponsors.

Current departmental research in concurrent systems includes the formal foundations of concurrent computation, methodologies for the design of distributed systems, system interconnection networks and parallel architectures and algorithms. The department is also engaged in research on the design of high-performance packet-switching systems capable of a variety of applications (including voice, data and video). Of particular interest are candidates with backgrounds in communications systems architecture, communications software, or performance analysis. In the area of intelligent computer systems, present research projects include computer vision, image processing, visual programming, speech recognition, expert systems and deductive databases. Candidates with backgrounds in any of these areas are encouraged to apply.

Candidates may reply to:

Dr. Jerome R. Cox, Jr., Chairman
Department of Computer Science
Washington University
Campus Box 1045
St. Louis, Missouri 63130

Applications requested by February 1, 1987.
Washington University is an equal opportunity/affirmative action employer.

UNIVERSITY OF CALIFORNIA, RIVERSIDE

Applications are invited for a tenure-track or tenure position in Computer Science beginning Fall 1987. Candidates must have demonstrated excellence in research and teaching. Research specialties in all areas of Computer Science will be considered but we are particularly interested in research areas in Computer Systems or Computer Methodology and Applications. The position is open as to the level of appointment.

Applicants should send a curriculum vitae and see that at least three letters of recommendation are sent to:

Professor Theodore J. Barth, Chair
Computer Science Search Committee
Department of Mathematics and Computer Science
University of California
Riverside, CA 92521

University of California, Riverside, is an Affirmative Action/Equal Opportunity Employer.

**CALIFORNIA STATE POLYTECHNIC
UNIVERSITY, POMONA**
Computer Science Department

Computer Science: Three tenure track positions, Assistant or Associate Professor, specialty open, A.Y. beg. Sept. 1987. Ph.D. strongly preferred (required for tenure or promotion). M.S. in C.S. with teaching experience is the minimum requirement. Send resume, three letters of reference, completed application form, confirmation of highest degree by March 1, 1987. Search Committee, Computer Science Department, California State Poly University, Pomona, CA 91768 AA/EE0

Now it's our turn

Bridge Communications, located on the beautiful San Francisco peninsula, is making a name for itself as the premier source for LAN products and technologies. We've started where the computer industry left off...and the market has never been stronger for our sophisticated information management solutions. If you'd like to work independently yet share your ideas and findings with like-minded colleagues in the heart of Silicon Valley, consider these opportunities.

Software Engineer

Use your experience writing synchronous line drivers and your knowledge of LAN data link and network protocols to work on the development, release, and support of internetwork bridges. This position is an opportunity for a self-starting individual who can assume project coordination efforts. MSCS/MSEE preferred; must have 3-5 years of networking experience, knowledge of UNIX*, C, and 68000, and familiarity with logic analyzers and line monitors. Job code: IB.

Software Engineer

If you're the rare person with TCP/IP and X.25 background, this is an opportunity to take on the development, release, and support responsibilities of high-performance routers and gateways. MSEE/MSCS and 3-5 years in networking required. Should be familiar with 68000 microprocessor, C, and UNIX. Job code: RP.

Software Engineer

Become a member of the team working on the development and support of our TCP/IP communications servers. BSCS/BSEE required, plus at least 3 years networking experience including a knowledge of UNIX and C. Job code: TC.

We offer an excellent benefits package (including stock options) and the rewards of working in one of the SF Bay Area's top high-growth companies. To apply, please indicate appropriate job code and send your resume to: Human Resources, BRIDGE COMMUNICATIONS, INC., 2081 Stierlin Road, Mountain View, CA 94043. An equal opportunity employer. M/F/H/V.

*UNIX is a trademark of AT&T Bell Labs.

Bridge Communications Inc.

ADVERTISER INDEX

AI-87	42
Bridge Communications Inc.	143
Buscon/87-West	119
CACI	1
CHI + GI 1987	39
Compcon-87	4-8
Computer Book Club	37
Gold Hill Computers, Inc.	Cover IV
ICC-87	97
ICCAD-87	18
IDA	83
Inmac	123
Library of Computer and Information Sciences	16-17
Lockheed Missiles and Space Co.	54
Macmillan Publishing Company	52
MIT Press	53
NATO ASI	74
9th Annual International Conference on Software Engineering	110-113
Omega Engineering, Inc.	Cover II
Orlando-87 Tutorial Week	125
PCs Limited	142
Reifer Consultants, Inc.	52
Second IEEE Computer Society Workshop on Software Technology Transfer	Cover III
Sixth Symposium on Reliability in Distributed Software and Database Systems	98-99
Third Annual Expert Systems in Government Conference	84
Third International Conference on Data Engineering	28-29
TRW	109
Westex-87	66
Workshop on Distributed Information Systems	122
Classified Advertising	131-143

FOR DISPLAY ADVERTISING INFORMATION CONTACT

Southern California and Mountain States: Richard C. Faust Company, 24050 Madison Street, Suite 100, Torrance, CA 90505; (213) 373-9604.

Northern California and Pacific Northwest: Don Farris Company, 161 W. 25th Ave., #102B, San Mateo, CA 94403; (415) 349-2222.

Jack Vance, P.O. Box 3205, Saratoga, CA 95070; (408) 741-0354.

East Coast: Hart Associates, Inc., P.O. Box 339, 42 Lake Blvd., Matawan, NJ 07747; (201) 583-8500.

New England: Arpin Associates, P.O. Box 227, Weston, MA 02193; (617) 899-5613.

George Watts, III, 4 Conifer Dr., Wilbraham, MA 01095; (413) 596-4747.

Midwest: Thomas Knorr, Knorr MicroMedia, Inc., 333 North Michigan Ave., Chicago, IL 60601; (312) 726-2633.

Southwest: The House Company, 3000 Wesleyan, Suite 345, Houston, TX 77027; (713) 622-2868.

For production information, conference or classified advertising contact Carole L. Porter, COMPUTER Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-1140.

PRODUCT INDEX

	RS#	Page#
Adra Systems	55	124
AI-87	3	42
Apollo Computer	48	122
Autodesk	53	123
Arnet Corporation	80	127
Borland International	54	124
Brooktree Corp.	70	126
Centram	49	122
Chips and Technologies	71	126
Computer Book Club	2	37
Computer Innovations	45	121
Coyne Kalajian	50	123
Datavue	41	121
Everex Systems Inc.	86	127
Genoa Systems Corp.	72	126
GESPAC	87	127
Gold Hill Computers, Inc.	9	Cover IV
Harris	57	124
Hewlett-Packard	44	121
Honeywell	73	126
Hybrid Systems Corp.	74	126
Hyundai Electronics America	75	126
IDEAssociates, Inc.	84	127
Inmac	7	121
Intel	42	121
Kaypro	46	122
Macmillan Publishing Company	5	52
Micro CAD/CAM	43	121
Micro Engineering Solutions	52	123
Microtex	51	122
MIT Press	5	53
Motola	89	127
NEC	40	121
New Micro	47	122
Omega Engineering, Inc.	1	Cover II
Plessey Microsystems	82	127
Pyramid Technology	56	124
Quadram	88	127
Qualogy	81	127
Reifer Consultants, Inc.	4	52
Signetics Corp.	76	126
Speech Plus, Inc.	85	127
Standard Microsystems Corp.	83	127
Toshiba America, Inc.	77	126
Xilinx, Inc.	78	126



COMPUTER

READER
SERVICE

For further information on advertised products, new products, or literature and to specify levels of interest in *Computer's* articles and departments fill out one of these cards. Affix stamp and mail.

Please print or type your name and address.

Reader Service Card

COMPUTER

Void after 7/15/87

1/87

Name _____

Company _____

Title _____

Address _____

City _____ State _____

Zip _____ Telephone _____

Circle your level of interest in the following articles and departments:

	High	Medium	Low	Did not read
Guest Editor's Intro	101	102	103	104
Computer Architectures	105	106	107	108
Software Development	109	110	111	112
Symbolics Architecture	113	114	115	116
FAIM-1	117	118	119	120
Smalltalk	121	122	123	124
DADO2	125	126	127	128
Connection Machine	129	130	131	132
Connectionist for AI	133	134	135	136
New Products	137	138	139	140
Letters	141	142	143	144
Update	145	146	147	148

I need more information on the following:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96	97	98

Products purchased or specified:

	Hobby	Job
Computers	150	151
Peripherals	152	153
Data Communication	154	155
Memories/components	156	157
Software and Services	158	159
Publications	160	161

- 170 Please send me the IEEE-CS 1986 Publications catalog.
- 171 Please send me an application for advancement to senior member.
- 172 Please send me an IEEE-CS membership application.

Reader Service Card

COMPUTER

Void after 7/15/87

1/87

Name _____

Company _____

Title _____

Address _____

City _____ State _____

Zip _____ Telephone _____

Circle your level of interest in the following articles and departments:

	High	Medium	Low	Did not read
Guest Editor's Intro	101	102	103	104
Computer Architectures	105	106	107	108
Software Development	109	110	111	112
Symbolics Architecture	113	114	115	116
FAIM-1	117	118	119	120
Smalltalk	121	122	123	124
DADO2	125	126	127	128
Connection Machine	129	130	131	132
Connectionist for AI	133	134	135	136
New Products	137	138	139	140
Letters	141	142	143	144
Update	145	146	147	148

I need more information on the following:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28
29	30	31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96	97	98

Products purchased or specified:

	Hobby	Job
Computers	150	151
Peripherals	152	153
Data Communication	154	155
Memories/components	156	157
Software and Services	158	159
Publications	160	161

- 170 Please send me the IEEE-CS 1986 Publications catalog.
- 171 Please send me an application for advancement to senior member.
- 172 Please send me an IEEE-CS membership application.

Comments on articles and other editorial matter

What did you find appealing about the articles/departments you liked? _____

What other topics/departments would you like to see?

General comments: _____

For reader service, see other side

This PO box for reader service cards only.

PLACE
STAMP
HERE

COMPUTER

magazine

Reader Service Inquiries
Box 24008
Los Angeles, Ca 90024
USA

Comments on articles and other editorial matter

What did you find appealing about the articles/departments you liked? _____

What other topics/departments would you like to see?

General comments: _____

For reader service, see other side

This PO box for reader service cards only.

PLACE
STAMP
HERE

COMPUTER

magazine

Reader Service Inquiries
Box 24008
Los Angeles, Ca 90024
USA

CALL FOR PAPERS

Second IEEE Computer Society Workshop on Software Technology Transfer

*June 10-12, 1987
Santa Fe, New Mexico*

This workshop will provide a forum for the discussion and exchange of methods and models of technology transfer. Topics to be addressed include:

- models of the transfer process,
- means of cooperation between research and development,
- measurement of the transfer process,
- the role of education and training, and
- psychological and social issues.

This workshop will be limited to 30-40 participants. Attendance will be determined on the basis of position papers. Interested parties should submit five (5) copies of a 3- to 5-page position paper no later than February 15, 1987 to:

Charles Richter
Software Technology Program
MCC
P.O. Box 200195
Austin, Texas 78720

Organizing Committee:

J. C. Browne, University of Texas at Austin
Kate Ehrlich, Symbolics
Charles Richter, MCC (General Chair)
Bill Riddle, Software Productivity Consortium
Joseph Urban, University of Southwestern Louisiana

Local Arrangements:

James Babcock, MCC



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.



THE COMPUTER SOCIETY
OF THE IEEE

TRANSFORM YOUR PC INTO A LISP MACHINE.



Not long ago, a specialized LISP machine was your only choice for serious AI development and delivery.

But now you can get LISP machine performance out of that ordinary IBM PC* sitting on your desk. With the **Gold Hill 386 LISP System**.

You simply plug in the System's HummingBoard™—unique 386-based hardware designed specifically for the LISP environment. Then you add the System's Golden Common LISP 386 Developer software.

That's all you need to transform your PC into a LISP machine. You can develop and deliver AI applications on your PC. And create your own expert systems. You'll get the kind of LISP performance you thought was only possible in a system costing ten times as much.

And if you don't need the whole system, Gold Hill can still offer you AI solutions. You can get GCLISP 286 Developer software for your PC.* And GCLISP 386 Developer software will be available for leading manufacturers' 386-based PCs.

Tomorrow's AI development tools for PCs are available today from Gold Hill. We'll prove it to you—just call to get the latest Gabriel Performance Benchmarks. You'll be amazed to learn what your PC is capable of. Call toll-free:

Gold Hill. The expert in AI on PCs.

1-800-242-LISP

*The Gold Hill 386 LISP System requires an IBM PC XT, AT or compatible. GCLISP 286 Developer Software requires an IBM PC AT or compatible.

© 1986 Gold Hill Computers, Inc. Gold Hill, Gold Hill 386 LISP System, Golden Common LISP, GCLISP, and Developer are trademarks of Gold Hill Computers, Inc. IBM PC, XT and AT are registered trademarks of International Business Machines Corp. HummingBoard is a trademark of A.I. Architects, Inc.



In Mass.: (617) 492-2071
Gold Hill Computers, Inc.
163 Harvard St.,
Cambridge, MA 02139